VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Integrace HTML5 do systému pro správu obsahu Typo

Integration of HTML5 into Content Management System Typo

2010 Tibor Piňo

Súhlasím so zverejnením tejto diplomovej práce podľa požiadaviek čl. 26, odst. 9 *Študijného a skúškového poriadku pre štúdium v magisterských programoch VŠB-TU Ostrava*.

V Ostrave 3. máj 2010 .....

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne zdroje a publikácie z ktorých som čerpal.

V Ostrave 3. máj 2010 .....

Ďakujem pánovi Ing. Marekovi Běhálkovi Ph.D. za ochotnú pomoc, odborné vedenie a podporu pri tvorbe mojej diplomovej práce.

Abstrakt

Cieľom diplomovej práce je integrácia HTML5 do systému pre správu obsahu Typo. Práca najprv zoznamuje čitateľa s použitými technológiami a to predovšetkým Ruby, Ruby on Rails, JavascriptMVC ako i so samotným systémom Typo. Ďalej čitateľovi približuje nový štandard HTML5 predstavením nových vlastností a technológií, ktoré prináša. Následne identifikuje všeobecné problémy webových stránok, ktoré sú vytvorené pomocou HTML 4 a ponúka návrh riešení prostredníctvom HTML5. Tu práca taktiež porovnáva v súčasnosti používané technológie s technológiami v HTML5. Nakoniec popisuje proces implementácie nových vlastností do systému Typo.

Kľúčové slová: HTML5, Typo, systém pre správu obsahu, Ruby on Rails, JavascriptMVC, SQLite, Sémantika, SVG, Model View Controller

Abstract

The main objective of this diploma work is an integration of HTML5 into content management system Typo. Firstly, I'm introducing already used technologies, mainly Ruby, Ruby on Rails, JavascriptMVC and Typo system itself. In the next part, I'm explaining the new HTML5 standards by showing the new properties and technologies it brings. Afterwards I'm identifying issues with currently used web sites and I'm offering solutions to resolve these issues using HTML5. In this part, I also compare currently used technologies with HTML5 technologies. In the final part I'm presenting the process of implementing these new properties into Typo system.

Keywords: HTML5, Typo, Content management system, Ruby on Rails, JavascriptMVC, SQLite, Semantics, SVG, Model View Controller

Zoznam použitých skratiek a symbolov

AJAX	–	Asynchronous JavaScript and XML
API	–	Application programming interface
CERN	–	European organization for nuclear research
CMS	–	Content management system
DBMS	–	Database management system
DOM	–	Document object model
HTML	–	Hyper text markup language
HTTP	–	Hypertext Transfer Protocol
MVC	–	Model View Controller
ORM	–	Object-relational mapping
OWL	–	Ontology Web Language
RIA	–	Rich internet application
RDF	–	Resource Description Framework
REST	–	Representational state transfer
URL	–	Uniform Resource Locator
URI	–	Uniform Resource Identifier
WHATWG	–	Web hypertext application technology working group
WYSIWYG	–	What you dee is what you het
WWW	–	World Wide Web
W3C	–	World Wide Web Consortium

Obsah

1 Úvod

V súčasnosti sa asi málokto z nás nestretol s pojmom Internet a WWW. Internet a samotné webové stránky zažívajú v posledných rokoch obrovský rozmach, o čom svedčí čoraz väčšie množstvo nových technológií a možností, ktoré samotné webové stránky ponúkajú. Značkovací jazyk HTML tvorí základ webových stránok. Posledná štandardizácia tohoto jazyka pochádza z roku 1999, z čoho vyplýva, že tento štandard je už zastaralý a nevyhovuje súčasným požiadavkám. Túto skutočnosť sa snaží zmeniť nový návrh štandardu s názvom HTML5, ktorému je venovaná táto diplomová práca.

Diplomová práca zoznamuje čitateľa s novými vlastnosťami a technológiami v HTML5, analyzuje a porovnáva HTML5 s technológiami, ktoré sú používané v súčasnosti a popisuje proces implementácie vybraných častí HTML5 do systému pre správu obsahu Typo. Celý tento proces je bližšie rozpísaný v jednotlivých kapitolách.

Kapitola 2 Systémy pre správu obsahu zoznamuje čitateľa so systémami pre správu obsahu predstavením ich vlastností. Porovnáva systém Typo so systémami, ktoré sa používajú v súčasnosti.

Kapitola 3 Technológie popisuje technológie, ktoré boli použité pri implementácii. Zameriava sa pritom na jazyk Ruby a vývojové rámce Ruby on Rails a JavascriptMVC.

Kapitola 4 HTML5 približuje čitateľovi nové vlastnosti a technológie v HTML5, kde čitateľa zoznamuje s novými elementami jazyka HTML ako i technológiami akými sú offline mód, Web Database API a iné.

Kapitola 5 Nové vlastnosti systému Typo bližšie špecifikuje nové implementované vlastnosti do tohto systému. Identifikuje problémy systému Typo, navrhuje riešenia týchto problémov prostredníctvom HTML5 a porovnáva navrhnuté riešenia so súčasnými technológiami.

Kapitola 6 Implementácia offline módu popisuje proces implementácie, ktorý je rozdelený do niekoľkých etáp. Jedná sa o špecifikáciu zadania, špecifikáciu požiadaviek, špecifikáciu chovania, analýzu, návrh a implementáciu.

Kapitola 7 Implementácia sémantiky popisuje proces implementácie sémantiky do systému Typo. Porovnáva prístupy, ktoré sú použité v tomto systéme s prístupmi, ktoré ponúka HTML5.

Kapitola 8 Implementácia SVG popisuje proces implementácie SVG do systému Typo. Zoznamuje čitateľa so špecifickými požiadavkami, ktoré sú kladené na mobilné webové stránky a predstavuje riešenia, ktoré nám ponúka SVG v spolupráci s HTML5.

2 Systémy pre správu obsahu

Skratka anglického slova content management system (systém pre správu obsahu alebo redakčný systém) je označenie pre softvér zaisťujúci správu dokumentov. Redakčný systém napomáha k organizácii, kontrole a publikovaniu veľkého množstva dokumentov i iného obsahu, ako sú obrázky alebo multimediálne zdroje. Takéto systémy obsahujú aj ďalšie rozšírenia na zjednodušenie činností spojených s publikovaním obsahu na webových stránkach.

2.1 Vlastnosti systémov pre správu obsahu

V súčasnosti existuje široké množstvo rôznych systémov pre správu obsahu. Systémy sa navzájom odlišujú svojou licenciou, funkčnosťou, množstvom rozšírení, podporou databáz, rozšíriteľnosťou a pod. Vo všeobecnosti však platí, že každý z takýchto systémov obsahuje určitú množinu spoločných vlastností. Súčasné systémy pre správu obsahu sa vyznačujú nasledovnými základnými vlastnosťami:

- vytváranie, modifikácia a úprava dokumentov, spravidla prostredníctvom webového rozhrania s využitím WYSIWYG editorov, alebo iných formátovacích nástrojov tak, aby nebola potrebná znalosť jazyka HTML,
- správa súborov,
- riadenie prístupu k dokumentom, definovanie jednotlivých rolí užívateľom,
- správa diskusií, komentárov,
- základné nastavenia šablóny stránky,
- možnosť jednoduchého rozšírenia pomocou rozšírení.

Medzi najznámejšie systémy pre správu obsahu v súčasnosti patria Wordpress [22], Joomla [24] a Drupal [23], ktoré sú založené na programovacom jazyku PHP alebo systémy Typo [19], Radiant [20] a Mephisto [21], ktoré sú založené na programovacom jazyku Ruby.

2.2 Systém pre správu obsahu Typo

V mojej práci som sa pri hľadaní vhodného systému pre správu obsahu, ktorý by som mohol použiť ako vývojový rámec pre svoju diplomovú prácu, zameral na systémy, ktoré sú založené na vývojom rámci Ruby on Rails a programovacom jazyku Ruby. Systémov, ktoré spĺňajú tieto požiadavky, je celá rada. Preto som sa pri výbere vhodného systému riadil podľa nasledovných kritérií:

- systém založený na vývojom rámci Ruby on Rails a programovacom jazyku Ruby,
- slobodná licencia,

- možnosť rozšíriť systém o nové vlastnosti,
- stabilný a aktívny projekt.

Uvedené kritéria najlepšie spĺňa systém pre správu obsahu Typo [19], ktorý si postupne predstavíme z pohľadu jeho vlastností.

Typo je systém pre správu obsahu, ktorý je založený na vývojom rámci Ruby on Rails a predstavuje jeden z najstarších systémov pre správu obsahu založených na tomto vývojovom rámci. Prvá verzia bola vydaná v roku 2001 a jej autorom bol Tobias Lütke. Za tento čas sa Typo vypracoval na stabilný projekt s množstvom vlastností. Medzi základné vlastnosti Typo patria:

- podpora tém,
- XML-RPC služby,
- SEO optimalizácia,
- podpora databáz MySQL, SQLite a PostgreSQL,
- rozšíriteľnosť systému pomocou pluginov,
- podpora pre viacerých užívateľov,
- anti-spam ochrana pred vkladaním nevyžiadaných komentárov.

2.3 Typo verzus iné systémy

Ako sme už spomínali, v súčasnosti existuje veľké množstvo rôznych systémov pre správu obsahu. Medzi najpoužívanejšie systémy patria systémy, ktoré sú postavené na programovacom jazyku PHP, a sú to napríklad Wordpress, Joomla alebo Drupal. Z pohľadu programátora je programovací jazyk najdôležitejším kritériom pri porovnávaní takýchto systémov, preto sa pri porovnaní týchto systémov zameriame práve na programovací jazyk. Systém Typo je postavený na programovacom jazyku Ruby a vývojovom rámci Ruby on Rails. Jazyk Ruby predstavuje plne objektovo orientovaný jazyk a systém Typo ponúka možnosť rozšíriť svoju funkcionalitu práve v tomto jazyku. Systém Typo je taktiež postavený na vývojom rámci Ruby on Rails, ktorý programátorovi ponúka množstvo užitočných nástrojov a prostriedkov. Programátor môže využiť MVC architektúru, Active Record, podporu rôznych databázových systémov atď. Na druhej strane stoja veľmi populárne a používané systémy, ktoré sú postavené na programovacom jazyku PHP. PHP nepredstavuje plne objektovo orientovaný jazyk a tieto systémy väčšinou poskytujú len veľmi nízku úroveň objektovo orientovaného programovania. Ďalším rozdielom je podpora rôznych databázových systémov, kde väčšinou takéto systémy podporujú len jeden databázový systém. Ďalším nedostatkom takýchto systémov je samotná architektúra. Keďže je Typo postavené na architektúre MVC, je vždy presne určené, akým spôsobom a kde sú dané časti systému štrukturované a vykonávané. Takéto konštatovanie neplatí pre tieto systémy a každý z týchto systémov používa vlastnú

štruktúru a spôsob vykonávania zdrojového kódu. Ako záver tohto porovnania môžeme konštatovať, že z pohľadu programátora práve jazyk Ruby a vývojový rámec Ruby on Rails posúvajú systém Typo pred ostatné systémy.

3 Technológie

V diplomovej práci je použitých množstvo technológií. Niektoré technológie sú z pohľadu implementácie významné viac, iné sú významné menej. My sa z tohto dôvodu zameriame len na vybrané technológie, ktoré sú z pohľadu implementácie najdôležitejšie.

3.1 Ruby

Ruby predstavuje objektovo orientovaný programovací jazyk, ktorého hlavným cieľom je jednoduchosť a produktivita. Ruby vychádza zo základného princípu, že programovanie je tvorivá činnosť, ktorá nám má prinášať radosť. Svojou syntaxou sa Ruby inšpiroval v jazykoch ako sú Perl, SmallTalk, Eiffel, Ada a Lisp. Tvorcom jazyka je Yukihiro Matsumoto, skôr známy pod prezývkou Matz. Ten ako zástanca objektovo orientovaného programovania hľadal v prvej polovici 90. rokov skriptovací jazyk, ktorý by vyhovoval jeho predstavám. Keďže takýto jazyk pôvodne nenašiel, rozhodol sa, že vytvorí vlastný jazyk. Vtedajšie rozhodnutie neskôr komentoval slovami: "Chcel som skriptovací jazyk, ktorý by bol viac výkonný než Perl a viac objektovo orientovaný než Python. Preto som sa rozhodol navrhnúť svoj vlastný jazyk [3]." Práce na novom jazyku sa začali v roku 1993 a prvá verzia bola uverejnená v roku 1995.

Ako sme už spomínali, hlavným cieľom Ruby je jednoduchosť a produktivita. Vďaka tomu má Ruby veľmi jednoduchú syntaxu, ktorá sa snaží priblížiť prirodzenému jazyku, a preto je kód veľmi čistý a ľahko čitateľný. Najlepšie si to môžeme ukázať na výpise zdrojového kódu 1.

```
3.times { print "Hello World!" } => Hello World!Hello World!Hello World!
```

Výpis 1: Ukážkový kód v jazyku Ruby

I keď vidíme syntaxu jazyka po prvýkrát, je nám na prvý pohľad jasné, čo bude výsledkom daného výpisu, pretože kód môžeme čítať ako bežnú reč.

Medzi základné vlastnosti jazyka Ruby patria:

- objektovo orientovaný jazyk,
- dynamické typovanie,
- plná podpora Unicode,
- interactive Ruby Shell,
- automatická správa pamäte,
- preťažovanie operátorov,
- centralizovaný správca balíčkov RubyGems.

Jazyk Ruby je vydaný pod slobodnou open source licenciou a dnes sa na jeho vývoji okrem samotného autora podieľa aj rozsiahla komunita vývojárov. K jeho veľkej popularite prispel aj samotný vývojový rámec Ruby on Rails, ktorý si predstavíme v nasledujúcej kapitole.

3.2 Ruby on Rails

Ruby on Rails je vývojový rámec pre vývoj webových aplikácií založený na jazyku Ruby. Jeho hlavným zameraním je agilný spôsob vývoja, ktorý je používaný pre rýchly vývoj webových aplikácií. Vďaka tomuto prístupu predstavuje Ruby on Rails unikátnu vývojovú platformu, ktorá umožňuje v oveľa kratšom čase vytvoriť plnohodnotné webové aplikácie. Toto je dosiahnuté základnými princípmi, na ktorých je Ruby on Rails postavený.

Don't repeat yourself (Neopakuj sám seba) - tento princíp môžeme chápať tak, že každý kúsok zdrojového kódu môžeme nájsť a je napísaný práve na jednom mieste. K tomuto účelu nám pomáhajú rôzne pomocné metódy tzv. helpers alebo partial views, ktoré nám umožňujú napísať kód jedenkrát a následne tento kód používať v celej našej aplikácii, bez nutnosti jeho ďalšieho duplikovania [5].

Convention over configuration (Konvencia má prednosť pred konfiguráciou) - mnoho aplikácií si je veľmi podobných. Načítavajú sa dáta z databázy, ukladajú sa do dátovej štruktúry, prevádzajú sa operácie nad týmito dátami a následne sa zobrazujú pomocou HTML. Podobne to funguje aj opačným smerom, kde sa získavajú dáta od užívateľa prostredníctvom formulára, následne sa transformujú a ukladajú do databázy. Problém je v tom, že málokomu sa chce každý nový projekt začínať vždy od samotného začiatku konfigurovaním pripojenia k databáze a následného písania *SELECT * FROM table* atď. Preto sa Ruby on Rails riadi filozofiou Convention over configuration, kde platí, že programátor konfiguruje iba to, čo sa líši od bežného a zaužívaného nastavenia. Dobrou ukážkou je situácia, kde máme model Article a aplikácia automaticky bez explicitného nastavovania hľadá dáta v tabuľke Articles, teda v tabuľke, ktorej názov predstavuje množné číslo názvu modelu.

Medzi základné vlastnosti Ruby on Rails patria:

- MVC architektúra,
- Active Record,
- ORM,
- databázové migrácie,
- podpora pluginov,
- vstavaná validácia dát.

Samotný vývojový rámec je zložený z niekoľkých balíčkov, ktoré vytvárajú ucelený celok. Medzi tieto balíčky patria ActiveRecord, ActiveResource, ActionPack, ActiveSupport a ActionMailer. My si bližšie predstavíme ActiveRecord, ktorý predstavuje jednu z najzákladnejších a najzaujímavejších častí Ruby on Rails.

ActiveRecord je srdcom každej Ruby on Rails aplikácie a dokonale demonštruje princípy, na ktorých je vývojový rámec postavený. ActiveRecord predstavuje dátovú vrstvu a funguje na základe objektovo-relačného mapovania ORM, tzn. že databázové tabuľky sú mapované na jednotlivé triedy. Predstavme si, že v databáze sa nachádza tabuľka orders a náš program bude obsahovať podľa konvencií triedu Order. Jednotlivé riadky tabuľky budú predstavovať objekty tejto triedy a jednotlivé stĺpce budú predstavovať atribúty objektu tejto triedy. Navyše ActiveRecord obsahuje niekoľko metód, ktoré nám umožňujú vykonávať top level operácie nad dátami. Predstavme si situáciu, kde by sme chceli nájsť objednávku, ktorej identifikačné číslo je rovné hodnote 1. Ukážku môžeme vidieť na výpise zdrojového kódu 2.

```
Order.find(1)
```

Výpis 2: Ukážka použitia metódy find()

Výsledkom uvedeného volania bude objekt, ktorý predstavuje riadok tabuľky s id, ktoré má hodnotu 1 a jednotlivé stĺpce databázovej tabuľky predstavujú atribúty daného objektu.

Na výpise zdrojového kódu 3 môžeme vidieť ďalšie použitie metódy find().

```
Order.find(:all)
```

Výpis 3: Výpis všetkých objednávok pomocou metódy find()

ActiveRecord nám taktiež umožňuje manipulovať s dátami. Ukážku môžeme vidieť na výpise zdrojového kódu 4.

```
order = Order.find(1)
order.status = ''sent''
order.save
```

Výpis 4: Zmena obsahu atribútu v tabuľke

Ako je nám známe z návrhu databáz, databázové tabuľky majú medzi sebou rôzne väzby. Sú to napríklad väzby 1:1, 1:N alebo M:N. Keďže modely sú mapované na jednotlivé databázové tabuľky, ActiveRecord umožňuje definovať tieto väzby priamo v modeloch. Rozšírme našu ukážku modelu Order o väzbu na užívateľa, kde platí, že objednávka má práve jedného užívateľa, avšak užívateľ môže mať N objednávok. Zvolíme preto väzbu medzi tabuľkami orders a users ako 1:N. Ukážku môžeme vidieť na výpise zdrojového kódu 5.

```
class Order < ActiveRecord::Base
  belongs to :user
end

class user < ActiveRecord::Base
  has many :users
end
```

Výpis 5: Väzby medzi tabuľkami definované pomocou modelu

Pre ďalšie zoznámenie s týmto vývojovým rámcom odporúčam celosvetovo uznávanú knihu Agile Web Development With Rails [5].

3.3 JavascriptMVC

JavascriptMVC je javascriptový vývojový rámec pre vývoj RIA aplikácií. RIA aplikácie môžeme chápať ako webové aplikácie, ktoré sa svojím chovaním a funkcionalitou podobajú na desktopové aplikácie [6]. Medzi základné vlastnosti vývojového rámca patrí architektúra MVC, ktorá ho odlišuje od vývojových rámcov podobného typu. JavascriptMVC nie je závislý na žiadnych serverových komponentách a môže spolupracovať s ľubovoľnou webovou službou alebo programovacím jazykom akými sú napríklad Ruby, PHP, Perl, ASP.NET, Python alebo Java.

JavascriptMVC je svojimi princípmi veľmi podobný Ruby on Rails, ktorý sme si predstavili v kapitole 3.2. Podobnosť môžeme nájsť nielen v MVC architektúre, ale aj v princípe Convention over configuration.

Medzi základné vlastnosti JavascripMVC patria:

- MVC architektúra,
- testovacie nástroje,
- nástroje pre tvorbu dokumentácie,
- správa chýb prostredníctvom DamnIT,
- Ajax a DOM funkcionalita prostredníctvom jQuery,
- reťazenie a kompresia kódu.

V diplomovej práci je prostredníctvom JavascriptMVC naprogramovaná offline časť systému Typo.

4 HTML5

HTML5 označuje názov ďalšej hlavnej revízie jazyka HTML, ktorý predstavuje základný značkovací jazyk pre WWW. Cieľom HTML5 je nahradiť existujúce a v súčasnosti používané štandardy HTML 4, XHTML 1.0 a DOM Level 2 HTML. Jeho hlavným cieľom je odstrániť nedostatky v týchto štandardoch, implementovať nové požiadavky, ktoré vznikli vývojom nových technológií a zredukovať nutnosť používania rôznych proprietárnych technológií akými sú Adobe Flash, Microsoft Silverlight alebo Sun JavaFX. Za týmto účelom HTML5 prináša množstvo nových vlastností a technológií, ktoré budú náplňou tejto kapitoly.

4.1 História

História HTML5 siaha do roku 2004, kedy sa sformovala pracovná skupina s názvom Web Hypertext Application Technology Working Group skrátene WHATWG. Podnetom k vzniku tejto skupiny bol pomalý vývoj webových štandardov riadených konzorciom W3C a snaha konzorcia nahradiť HTML technológiami založenými na XML. Po jej vzniku začala skupina pracovať na návrhu nového štandardu, ktorý zahŕňal dokumenty s názvami Web Forms 2.0, Web Apps 1.0 a Web Controls 1.0 [8]. V roku 2007 vzniká nová pracovná skupina pod vedením W3C [9], ktorej cieľom je vytvoriť nový štandard jazyka HTML a ako základ pre nový štandard je použitý pracovný návrh skupiny WHATWG. Budúci štandard dostáva názov HTML 5, neskôr je názov upravený na HTML5. Prvá verejne dostupná pracovná verzia vychádza v roku 2008. Všetky doterajšie výsledky a budúce ciele môžeme zhrnúť do nasledujúcej časovej postupnosti [1].

- Máj 2007 - HTML5 and Web Forms 2.0 špecifikácie adoptované ako základ pre budúci štandard.
- November 2007 - HTML Design Principles First Public Working Draft.
- Február 2008 - HTML5 First Public Working Draft.
- Január 2010 - HTML5 Last Call Working Draft.
- December 2010? - HTML5 Candidate Recommendation.
- Január 2012? - HTML5 Proposed Recommendation.
- Marec 2012? - HTML5 Recommendation.

V čase písania tohto textu sa špecifikácia HTML5 nachádza v stave Last Call, čo je stav, kedy sa prijímajú už len návrhy na opravy chýb a špecifikácia by sa nemala výrazne meniť.

[1] z dôvodu ponechania pôvodných stavov špecifikácie sú výrazy uvedené v anglickom jazyku

4.2 Syntax

HTML5 definuje syntaxu, ktorá je kompatibilná s HTML 4 a zároveň s XHTML 1 dokumentami. Dokumenty používajúce HTML syntaxu musia byť servírované s hodnotou media type *text/html*. Na výpise zdrojového kódu 6 môžeme vidieť dokument, ktorý je v súlade s HTML syntaxou.

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF−8">
        <title>Example document</title>
    </head>
    <body>
        <p>Example paragraph</p>
    </body>
</html>
```

Výpis 6: Ukážka dokumentu používajúceho HTML syntaxu

Ďalšia syntax, ktorá môže byť použitá v jazyku HTML5 je syntax XML. Táto syntax je kompatibilná s XHTML 1 dokumentami. Dokumenty používajúce XML syntaxu musia byť servírované s hodnotou media type *application/xhtml+xml* alebo *application/xml*, a elementy musia byť z menného priestoru http://www.w3.org/1999/xhtml, a dodržiavať pravidlá podľa XML špecifikácie [12]. Na výpise zdrojového kódu 7 môžeme vidieť dokument, ktorý je v súlade s XML syntaxou.

```
<?xml version="1.0" encoding="UTF−8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Example document</title>
    </head>
    <body>
        <p>Example paragraph</p>
    </body>
</html>
```

Výpis 7: Ukážka dokumentu používajúceho XHTML syntaxu

4.2.1 Kódovanie znakov

Pre HTML syntaxu v HTML5 máme tri možnosti, ako definovať kódovanie znakov. Medzi tieto možnosti patria:

- použitie HTTP hlavičky Content-Type,
- použitie Unicode Byte Order Mark na začiatku súboru, ktorý poskytuje podpis pre kódovanie súboru,

- použitie *meta* elementu s atribútom *charset*.

HTML5 zavádza novú zjednodušenú syntaxu pre kódovanie znakov prostredníctvom *meta* elementu. Ak by sme chceli nastaviť kódovanie znakov na hodnotu UTF-8, náš zápis by vyzeral ako na výpise zdrojového kódu 8.

```
<meta charset="UTF−8">
```

Výpis 8: Kódovanie znakov pomocou meta elementu v jazyku HTML5

Keďže HTML syntax jazyka HTML5 je spätne kompatibilná s predchádzajúcou syntaxou, stále môžeme použiť predchádzajúci spôsob zápisu. Tento zápis môžeme vidieť na výpise zdrojového kódu 9.

```
<meta http−equiv="Content−Type" content="text/html;charset=UTF−8">
```

Výpis 9: Kódovanie znakov pomocou meta elementu v jazyku HTML 4

Pre XHTML syntaxu v HTML5 musíme použiť spôsob zápisu aký nám určuje špecifikácia XML [12]. Spôsob zápisu kódovania môžeme vidieť na výpise zdrojového kódu 7.

4.2.2 DOCTYPE

HTML syntax jazyka HTML5 vyžaduje špecifikovanie DOCTYPE, aby bolo zaistené, že prehliadač bude spracovávať webovú stránku v štandardnom móde. DOCTYPE nemá žiadny iný význam, preto je pri použití XML syntaxe nepovinný a to z dôvodu, že takéto dokumenty sú vždy spracovávané v štandardnom móde. Na výpise zdrojového kódu 10 môžeme vidieť, akým spôsobom definujeme DOCTYPE v HTML5.

```
<!DOCTYPE html>
```

Výpis 10: Definovanie DOCTYPE v jazyku HTML5

Zápis DOCTYPE v predchádzajúcich verziách HTML bol oveľa dlhší a to z dôvodu, že jazyk bol založený na metajazyku SGML a preto vyžadoval referenciu na DTD. V HTML5 je DOCTYPE potrebný len pre zapnutie štandardného módu pri spracovaní stránok v prehliadačoch, preto nie je tento zdĺhavý zápis potrebný.

4.3 MathML a SVG

HTML syntax jazyka HTML5 umožňuje použitie MathML a SVG elementov v rámci HTML dokumentu spôsobom, že tieto elementy môžu byť použité ako štandardné elementy jazyka HTML. Na výpise zdrojového kódu 11 môžeme vidieť použitie jazyka SVG v HTML.

```
<body>
<svg width="100%" height="100%" version="1.1">
    <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="red"/>
</svg>
</body>
```

Výpis 11: Ukážka vykreslenia kružnice pomocou SVG v HTML

4.4 Nové elementy jazyka HTML

Medzi najväčšie zmeny v HTML5 patrí samotný jazyk HTML. Do špecifikácie boli zaradené nové elementy, iné menej používané alebo nepotrebné elementy boli buď nahradené novšími alebo úplne odstránené.

WHATWG pri návrhu nových elementov vychádzala zo súčasných potrieb webových vývojárov a z nedostatkov, ktorými trpí posledná špecifikácia HTML 4. Hlavným cieľom nových elementov v HTML5 je priniesť sémantiku do tohto jazyka. Sémantika je náuka o význame jazykových jednotiek a v kontexte HTML ju môžeme chápať ako stav, kde každý obsah má svoj presne určený význam.

V nasledujúcich kapitolách si bližšie predstavíme nové elementy a uvedieme si príklady ich použitia.

4.4.1 Element <article>

Element *article* predstavuje kompozíciu, ktorá formuje nezávislú časť dokumentu, webovej stránky alebo aplikácie. Element *article* môže byť použitý ako príspevok vo fórume, článok, komentár, interaktívny widget alebo ako iná nezávislá časť obsahu [10]. Použitie elementu môžeme vidieť na výpise zdrojového kódu 12, kde je element použitý ako článok.

```
<article>
    <header>
        <h1>The Very First Rule of Life</h1>
        <p><time pubdate datetime="2009-10-09T14:28-08:00"></time></p>
    </header>
    <p>If there's a microphone anywhere near you, assume it's hot and sending whatever you're
        saying to the world. Seriously.</p>
</article>
```

Výpis 12: Ukážka použitia elementu <article>

4.4.2 Element <aside>

Element *aside* predstavuje sekciu stránky, ktorá sa skladá z obsahu, ktorý priamo súvisí s obsahom okolo elementu *aside*. Takéto sekcie sú často reprezentované ako sidebar v tlačenej typografii [10]. Element *aside* môže byť taktiež použitý pre citácie, sidebar pre

umiestnenie reklamy, zoskupenie navigačných elementov alebo pre ostatný obsah, ktorý môže byť použitý oddelene od hlavného obsahu stránky. Použitie elementu môžeme vidieť na výpise zdrojového kódu 13.

```
<body>
    <header>
        <h1>My wonderful blog</h1>
    </header>
    <aside>
        <nav>
            <h1>Archives</h1>
            <ol reversed>
                <li><a href="/last-post">My last post</a>
            </ol>
        </nav>
    </aside>
    <article>
        <h1>My first post</h1>
    </article>
</body>
```

Výpis 13: Ukážka použitia elementu <aside>

4.4.3 Element <audio>

Element *audio* predstavuje zvuk alebo zvukový tok. Spoločne s týmto elementom špecifikácia prináša aj javascript API, ktoré umožňuje pracovať so zvukom alebo zvukovým tokom, ktoré sú prehrávané prostredníctvom tohto elementu. Použitie elementu môžeme vidieť na výpise zdrojového kódu 14.

```
<audio src="elvis.ogg" controls autobuffer></audio>
```

Výpis 14: Ukážka použitia elementu <audio>

V prípade elementu *audio* je situácia okolo licencií a podporovaných kódekov podobná, ako v prípade elementu *video* 4.4.13. Zatiaľ nie je jasné, aké kódeky budú podporované štandardom.

4.4.4 Element <canvas>

HTML5 prostredníctvom elementu *canvas* poskytuje javascript API, ktoré umožňuje vykresľovanie bitmapovej grafiky. Element *canvas* môže byť použitý na vykresľovanie grafov, grafiky hier alebo iných vizuálnych obrázkov za behu aplikácie [11].

4.4.5 Element <footer>

Element *footer* predstavuje pätičku pre obsah najbližšej predchádzajúcej sekcie alebo pre koreňový element. Element *footer* typicky obsahuje informácie o sekcii, v ktorej

je umiestnený. Medzi takéto informácie napríklad patria informácie o autorovi autor, odkazy na súvisiace dokumenty, informácie o autorských právach a iné [10]. Treba však ešte poznamenať, že element *footer* nepatrí do elementov, ktoré predstavujú elementy vytvárajúce nové sekcie a preto jeho použitie neuvádza novú sekciu dokumentu. Použitie elementu môžeme vidieť na výpise zdrojového kódu 15.

```
<article>
    <h1>The Very First Rule of Life</h1>
    <p>If there's a microphone anywhere near you, assume it's hot and sending whatever you're
        saying to the world. Seriously.</p>
    <footer>
        Posted in  Articles ,  Comments 3
    </footer>
</article>
```

Výpis 15: Ukážka použitia elementu <footer> ako pätičky pre článok

Výpis zdrojového kódu 16 nám ukazuje použitie elementu *footer* ako pätičky pre koreňový element <body>.

```
<body>
    <hgroup>
        <h1>My wonderful blog</h1>
        <h2>Everything about HTML</h2>
    </hgroup>
    <p>If there's a microphone anywhere near you...</p>
    <footer>
        Copyright information
    </footer>
</body>
```

Výpis 16: Ukážka použitia elementu <footer> ako pätičky pre koreňový element

4.4.6 Element <header>

Element *header* reprezentuje skupinu uvádzajúcich alebo navigačných informácií. Element *header* typicky obsahuje hlavičku sekcie v podobe elementov *h1* až *h6* alebo *hgroup* element, avšak môže obsahovať aj ďalší obsah, napríklad dátum publikovania obsahu [10]. Použitie elementu ako hlavičky pre článok môžeme vidieť na výpise zdrojového kódu 17.

```
<article>
    <header>
        <h1>The Very First Rule of Life</h1>
        <p><time pubdate datetime="2009-10-10T19:10-08:00">2009-10-10
            19:10-08:00</time></p>
    </header>
```

```
    <p>If there's a microphone anywhere near you, assume it's hot and sending whatever you're
        saying to the world. Seriously.</p>
</article>
```

Výpis 17: Ukážka použitia elementu <header>

4.4.7 Element <hgroup>

Element *hgroup* reprezentuje hlavičku sekcie. Element *hgroup* je určený na zoskupenie hlavičkových elementov *h1* až *h6* v prípade, že hlavička má niekoľko nižších úrovní, ako sú hlavičky alebo alternatívne nadpisy [10]. Použitie elementu môžeme vidieť na výpise zdrojového kódu 18.

```
<hgroup>
    <h1>The reality dysfunction</h1>
    <h2>Space is not the only void</h2>
</hgroup>
```

Výpis 18: Ukážka použitia elementu <hgroup>

4.4.8 Element <input>

Element *input* reprezentuje užívateľský vstup prostredníctvom formulárového prvku. Každý element *input* obsahuje atribút *type*, ktorý určuje, o aký typ formulárového prvku sa jedná a aké hodnoty môže tento prvok nadobúdať. Jazyk HTML5 prostredníctvom nových hodnôt atribútu *type* prináša niekoľko nových formulárových prvkov do jazyka HTML.

- tel - reprezentuje vstup pre telefónne číslo.
- search - reprezentuje vstup pre text.
- url - reprezentuje vstup pre absolútnu URL.
- email - reprezentuje vstup pre zoznam emailových adries.
- datetime - reprezentuje vstup pre špecifický dátum a čas.
- date - reprezentuje vstup pre špecifický dátum.
- month - reprezentuje vstup pre špecifický mesiac.
- week - reprezentuje vstup pre špecifický týždeň.
- time - reprezentuje vstup pre špecifický čas.
- datetime-local - reprezentuje vstup pre lokálny dátum a čas.

- number - reprezentuje vstup pre text predstavujúci číslo.
- range - reprezentuje vstup pre text predstavujúci číslo z určitého rozsahu.
- color - reprezentuje vstup pre farbu.

4.4.9 Element <meter>

Element *meter* predstavuje skalárny rozmer v rámci uvádzaného rozsahu alebo čiastočnú hodnotu, napríklad použitie diskového priestoru alebo relevanciu sql výsledku [10]. Treba však poznamenať, že element *meter* nepredstavuje skalárnu hodnotu alebo ľubovoľný rozsah, napríklad je nesprávne používať element pre vyjadrenie váhy alebo výšky, pokiaľ nie je známa maximálna hodnota týchto hodnôt. Použitie elementu môžeme vidieť na výpise zdrojového kódu 19.

```
<p>Your score is:
<meter value="91" min="0" max="100" low="40" high="90" optimum="100">A+</meter></p>
```

Výpis 19: Ukážka použitia elementu <meter>

4.4.10 Element <nav>

Element *nav* predstavuje sekciu stránky, ktorá odkazuje na iné stránky alebo na časti v rámci jednej stránky pomocou odkazov. V tomto kontexte element *nav* môžeme chápať ako sekciu s navigačnými odkazmi. Treba však dodať, že nie všetky skupiny odkazov, ktoré sa nachádzajú na stránke, sú vhodné na použitie spoločne s elementom *nav*. S týmto elementom by mali byť použité len skupiny odkazov, ktoré predstavujú hlavnú navigáciu. Použitie elementu môžeme vidieť na výpise zdrojového kódu 20, kde sa nachádzajú skupiny odkazov, ktoré predstavujú hlavnú navigáciu, ale taktiež skupinu odkazov, ktorá nie je vhodná na použitie s elementom *nav*.

```
<body>
    <h1>The Wiki Center Of Exampland</h1>
    <nav>
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/events">Current Events</a></li>
        </ul>
    </nav>
    <article>
        <h1>Demos in Exampland</h1>
        <p>Article...</p>
    </article>
</body>
```

Výpis 20: Ukážka použitia elementu <nav>

4.4.11 Element <section>

Element *section* predstavuje všeobecnú sekciu alebo sekciu aplikácie. Element *section* môžeme chápať ako tematické zoskupenie obsahu, väčšinou uvádzané titulkom [10]. Špecifikácia ďalej dodáva, že element nepredstavuje všeobecný element, ktorý by mohol byť použitý ako tzv. kontajner pri vytváraní HTML. V prípade, že potrebujeme element kvôli špecifickému použitiu štýlov alebo javascriptu, mali by sme v takomto prípade použiť element *div* namiesto elementu *section*. V tomto prípade platí, že pokiaľ sekciu nie je možné prirodzene uviesť titulkom, použitie elementu *section* nie je vhodné. Použitie elementu môžeme vidieť na výpise zdrojového kódu 21.

```
<section>
  <h1>Red Delicious</h1>
  <p>These bright red apples are the most common found in many supermarkets.</p>
</section>
```

Výpis 21: Ukážka použitia elementu <section>

4.4.12 Element <time>

Element *time* reprezentuje čas v 24 hodinovom formáte alebo presný dátum v Gregoriánskom kalendári voliteľne doplnený o čas alebo časové pásmo [10]. Použitie elementu môžeme vidieť na výpise zdrojového kódu 22.

```
<time datetime="2009-11-13">13 November 2009</time>
<time datetime="2010-11-13">Bruce's 21st birthday</time>
<time datetime="2010-11-13T020:00Z">8PM on my birthday</time>
<time datetime="2010-11-13T020:00+09:00">8PM on my birthday in Tokyo</time>
```

Výpis 22: Ukážka použitia elementu <time>

Voliteľne môže element *time* obsahovať atribút *pubdate*. Atribút *pubdate* predstavuje boolean hodnotu. Ak je atribút špecifikovaný, označuje, že dátum a čas elementu predstavuje dátum a čas publikovania najbližšieho *article* predchodcu. Ak element nemá žiadneho *article* predchodcu, predstavuje dátum a čas publikovania dokumentu ako celku [10]. Použitie atribútu *pubdate* môžeme vidieť na výpise zdrojového kódu 23.

```
<article>
    <header>
        <h1>Come to my party on <time datetime="2010-03-01">1 March</time></h1>
        <p>Published on <time datetime="2010-01-20" pubdate>20 January 2010</time> in
            the Sexysocks category.</p>
    </header>
    <p>I'm throwing a sexy socks party at Dr Naughty's Ladyboy Cabaret Roller-disco Bierkeller
        Pizza-parlour-a-gogo! Do come.</p>
</article>
```

Výpis 23: Ukážka použitia atribútu *pubdate* elementu <time>

4.4.13 Element <video>

Element *video* predstavuje element, ktorý je určený na prehrávanie videa alebo filmov [10]. Špecifikácia je v definícii veľmi stručná, zato však veľmi výstižná. Cieľom elementu *video* je prehrávať video obsah bez nutnosti používania technológií tretích strán, napríklad Flash, Quicktime alebo Silverlight. Spoločne s týmto elementom špecifikácia prináša aj javascript API, ktoré umožňuje pracovať s videom, ktoré je prehrávané prostredníctvom tohto elementu. Použitie elementu môžeme vidieť na výpise zdrojového kódu 24.

```
<video src="http://www.youtube.com/v/oHg5SJYRHA0&hl=en&fs=1&" autoplay></video>
```

Výpis 24: Ukážka použitia elementu <video>

Ako sa však ukazuje, veľkou prekážkou štandardného použitia naprieč všetkými prehliadačmi je určenie jediného kódeku, ktorý by sa stal štandardom pre všetky kódované videá. Prekážkami sú rôzne licencie a záujmy rôznych technologických firiem. Zatiaľ nie je jasné, aké kódeky budú štandardom podporované, je však veľmi pravdepodobné, že ak budeme chcieť zabezpečiť prezeranie videa vo všetkých prehliadačoch, budeme musieť naše video kódovať viac než jedenkrát pomocou viacerých kódekov [11].

4.5 Ostatné nové elementy

HTML5 prináša ešte niekoľko nových elementov, ktoré však svojim zameraním výrazne neovplyvňujú používané elementy z jazyka HTML 4 a tieto elementy budú väčšinou aplikované len v špeciálnych situáciách. Medzi tieto elementy patria <command>, <datalist>, <embed>, <keygen>, <mark>, <output>, <progress>, <rp>, <rt>, <ruby>, <source> a <summary>. Podrobné informácie o týchto elementoch sa môžeme dozvedieť zo špecifikácie jazyka HTML5 [10].

4.6 Zmenené elementy

HTML5 okrem nových elementov prináša zmeny vo význame a používaní existujúcich elementov. Medzi elementy, ktorých povaha použitia sa líši od predchádzajúcej špecifikácie patria <b>, , <hr>, <menu>, <meta>, <new>, <small> a . Podrobné informácie o týchto elementoch sa môžeme dozvedieť zo špecifikácie jazyka HTML5 [10].

4.7 Offline API a offline webové aplikácie

Cieľom tohto rozšírenia jazyka HTML je umožniť užívateľom pracovať s webovou stránkou aj v prípade, že dočasne stratili internetové pripojenie. Celý mechanizmus je založený na manifest súbore, ktorý obsahuje zoznam súborov, ktoré sú potrebné, aby webová stránka mohla pracovať v offline móde. Prehliadače rovnako na základe manifest súboru ukladajú kópie týchto súborov do internej pamäte, aby boli k dispozícii aj bez

internetového pripojenia. Spoločne s offline mechanizmom špecifikácia prináša aj javascript API, ktoré umožňuje prácu s internou pamäťou prehliadača.

Mechanizmus práce s manifest súborom by sme mohli zhrnúť do niekoľkých postupných krokov. Ako prvý krok je vytvorenie samotného manifest súboru, v ktorom určíme, ktoré súbory si má internetový prehliadač uložiť do svojej internej pamäte. Následne v našich HTML súboroch vložíme odkaz na vytvorený manifest súbor. Ak užívateľ navštívi takúto stránku, internetový prehliadač si stiahne kópiu manifest súboru a taktiež kópie všetkých súborov uvedených v tomto súbore. Ak užívateľ neskôr stratí pripojenie a pokúsi sa zobraziť webovú stránku opäť, internetový prehliadač zobrazí webovú stránku z internej pamäte a užívateľ ani nespozná, že stratil pripojenie.

Formát manifest súboru nám ukazuje výpis zdrojového kódu 25, kde sme určili, že súbory clock.html, clock.css a clock.js sa majú uložiť do internej pamäte prehliadača a budú dostupné aj v prípade nedostupnosti pripojenia.

```
CACHE MANIFEST
clock.html
clock.css
clock.js
```

Výpis 25: Formát manifest súboru

Pre správne fungovanie celého mechanizmu potrebujeme v našich HTML súboroch uviesť odkaz na samotný manifest súbor. Ako by vyzeral HTML súbor, ktorý by odkazoval na manifest súbor ukazuje výpis zdrojového kódu 26.

```
<!DOCTYPE HTML>
<html manifest="clock.manifest">
    <head>
        <title>Clock</title>
        <script src="clock.js"></script>
        <link rel="stylesheet"  href="clock.css">
    </head>
    <body>
        <p>The time is: <output id="clock"></output></p>
    </body>
</html>
```

Výpis 26: Definovanie odkazu na manifest súbor

Pre správne fungovanie offline aplikácie je však ešte potrebné, aby manifest súbor bol posielaný ako typ súboru *text/cache-manifest*.

4.7.1 Web SQL Database

Webové stránky, ktoré nie sú len obyčajnými statickými stránkami, ukladajú dáta typicky na webovom serveri. Aby mohli takéto aplikácie pracovať aj v offline móde, musia mať tieto dáta k dispozícii lokálne. Cieľom Web SQL Database je poskytnúť dátové úložisko, prostredníctvom ktorého si budú môcť webové stránky ukladať svoje dáta v

offline móde. K tomuto účelu HTML5 poskytuje relačnú databázu SQLite a príslušné javascript API, prostredníctvom ktorého môžeme pracovať s touto databázou [2]. Na výpise zdrojového kódu 27 môžeme vidieť prácu s offline databázou.

```
var db = openDatabase("notes", "", "The Example Notes App!", 1048576);

db.transaction(function(tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS Notes(title TEXT, body TEXT)',
        []);
    tx.executeSql('SELECT * FROM Notes', [], function(tx, rs) {
        for(var i = 0; i < rs.rows.length; i++) {
            renderNote(rs.rows[i]);
        }
    });
});
```

Výpis 27: Práca s offline databázou v HTML5

4.7.2 Web Storage

Špecifikácia HTML5 zavádza dva nové mechanizmy, ktoré sú svojím princípom podobné mechanizmu COOKIES. Ich účelom je ukladať štrukturované dáta na strane klienta. Jedná sa o mechanizmy *localStorage* a *sessionStorage* [3]. Dáta sú ukladané vo formáte kľúč-hodnota.

localStorage - slúži na ukladanie štrukturovaných dát na strane klienta, kde sú tieto dáta zdieľané v rámci jednej domény. To znamená, že ak si v internetovom prehliadači otvoríme niekoľko inštancií jednej domény, na ktorej beží nejaká webová stránka, všetky tieto inštancie budú zdieľať rovnaké hodnoty uložené v *localStorage*. Použitie *localStorage* môžeme vidieť na výpise zdrojového kódu 28.

```
localStorage['key'] = "value";
```

Výpis 28: Ukážka použitia localStorage

sessionStorage - slúži na ukladanie štrukturovaných dát na strane klienta, kde sú tieto dáta jedinečné pre každú inštanciu domény. To znamená, že *sessionStorage* môžeme pokladať za opak *localStorage*, kde boli dáta zdieľané všetkými inštanciami jednej domény. Použitie *sessionStorage* môžeme vidieť na výpise zdrojového kódu 29.

```
sessionStorage['key'] = "value";
```

Výpis 29: Ukážka použitia sessionStorage

[3] Pre Web Storage bola vyčlenená vlastná špecifikácia [17], avšak WHATWG stále pracuje na špecifikácii HTML5 spoločne s Web Storage v rámci jedného procesu.

4.8 Drag and Drop API

Špecifikácia HTML5 prináša aj niekoľko nových API, prostredníctvom ktorých prináša novú funkcionalitu. Jedným z nich je aj natívne Drag and Drop API, ktoré prináša mechanizmus drag-and-drop [30], ktorý bežne poznáme z desktopového prostredia. Drag-and-drop mechanizmus je založený na udalostiach a môžeme ho definovať ako sled udalostí, kedy drag operácia môže byť počiatočnou akciou pre udalosť *mouse-down*, nasledovaná sériou *mousemove* udalostí a zakončená operáciou drop, ktorá môže byť počiatočnou akciou pre udalosť *mouseup*. Aby sme zaistili, že element môže byť ľubovoľne presúvaný v rámci dokumentu, HTML5 nám poskytuje atribút *dragable*, ktorý nadobúda hodnôt *true* alebo *false*. Jednoduchý príklad, kde by sme mohli presúvať jednotlivé položky zo zoznamu obrázkov do koša, ktorý reprezentuje element div s id drop, môžeme vidieť na výpise zdrojového kódu 30.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset=utf-8 />
        <title>Basic Drag and Drop</title>
    </head>
    <body>
        <img src="img1.jpg" alt="Image 1" />
        <img src="img2.jpg" alt="Image 2" />
        <img src="img3.jpg" alt="Image 3" />
        <div id="drop"></div>
        <script>function cancel(e) {
            if (e.preventDefault) {
                e.preventDefault();
            }
            return false;
        }

        var drop = document.querySelector('#drop');

        addEvent(drop, 'dragover', cancel);
        addEvent(drop, 'dragenter', cancel);

        addEvent(drop, 'drop', function (e) {
            if (e.preventDefault) e.preventDefault();

            this.innerHTML += '<p>' + e.dataTransfer.getData('Text') + '</p>';
            return false;
        });
        </script>
    </body>
</html>
```

Výpis 30: Ukážka použitia Drag and Drop API

4.9 Editovateľné elementy

HTML5 umožňuje taký stav elementov, kedy môže byť ich obsah editovateľný. To znamená, že užívateľ môže priamo v internetovom prehliadači meniť obsah jednotlivých elementov bez nutnosti práce s formulárovými prvkami. Za týmto účelom špecifikácia zavádza dva nové atribúty, ktorými sú *contenteditable* a *designMode*. Atribút *contenteditable* nadobúda hodnôt true alebo false a na základe tejto hodnoty určuje, či daný element má byť editovateľný alebo nie. Atribút *designMode* nadobúda hodnôt on alebo off a na základe tejto hodnoty určuje, či celý dokument má byť editovateľný alebo nie.

4.10 Ostatné zmeny a novinky v HTML5

HTML5 prináša celú radu zmien a noviniek. Mnohé z nich sú náplňou tejto diplomovej práce, mnohé však nie sú v tejto práci uvedené. Cieľom tejto diplomovej práce nie je ponúknuť kompletné predstavenie všetkých zmien a noviniek v HTML5, z tohto dôvodu pre ďalšie informácie odporúčam dokumenty HTML5 Draft standard [10] a HTML5 differences from HTML4 [14].

5 Nové vlastnosti systému Typo

Hlavným cieľom diplomovej práce je navrhnúť a implementovať nové vlastnosti jazyka HTML5 do systému pre správu obsahu Typo. Pred samotnou implementáciou nových vlastností musíme najprv identifikovať nedostatky tohto systému a na základe zistených nedostatkov nájsť vhodné riešenia, ktoré ponúka HTML5. V tejto kapitole si postupne identifikujeme jednotlivé nedostatky systému, navrhneme riešenia identifikovaných nedostatkov a porovnáme navrhnuté riešenia s riešeniami podobných technológií. Tieto identifikované nedostatky a problémy budeme chápať v kontexte systému Typo. Avšak systém Typo predstavuje v kontexte WWW a HTML bežnú webovú stránku, preto môžeme dané problémy zovšeobecniť na akúkoľvek webovú stránku a nepovažovať ich len za problémy tohto systému.

5.1 Sémantika

Systémy pre správu obsahu, ako už z ich názvu vypovedá, slúžia primárne pre správu obsahu. To znamená, že prostredníctvom takýchto systémov je zobrazované a spravované veľké množstvo dát a najrôznejších informácií v najrozličnejších formách. S prezentovaním informácií prostredníctvom jazyka HTML veľmi úzko súvisí sémantika, teda význam takýchto informácií. Veľkým nedostatkom súčasného štandardu HTML je absencia sémantiky v HTML kóde. Sémantiku v kontexte HTML môžeme chápať ako stav, kedy HTML kód má jasne danú štruktúru a význam. Sémantika umožňuje informáciám určeným pre užívateľa, aby mohli byť taktiež automaticky spracovávané softvérom. Na tomto mieste sa môžeme zamyslieť, že predsa HTML kód je automaticky spracovávaný softvérom, napríklad robotmi internetových vyhľadávačov, avšak takéto spracovávanie je obtiažne z dôvodu, že súčasný značkovací jazyk HTML neumožňuje pridať hlbší význam týmto dátam. Človek dokáže na prvý pohľad rozoznať význam zobrazovaných informácií, napríklad v prípade, že sa jedná o navigačné menu, článok a iné. Avšak pre softvér je takáto informácia len zreťazenie znakov bez hlbšieho významu. Môžeme však oponovať, že súčasný štandard jazyka HTML obsahuje sémantické elementy, akými sú napríklad <p>, <code> a iné. Áno, takéto tvrdenie je pravdivé, avšak skôr môžeme sémantiku v jazyku HTML 4 chápať ako veľmi úzky základ a pri súčasnom vývoji si s týmito elementami vystačíme len veľmi obmedzene. Preto jedným z hlavných cieľov HTML5 bolo určiť spôsob, ktorý by jasným a jednoznačným spôsobom umožňoval vývojárom pridať do jazyka HTML sémantiku.

Na tomto mieste sa môžeme zamyslieť aj nad rôznymi problémami, ktoré sú spojené so zavedením sémantiky do tohto jazyka. Jedným z týchto problémov je spätná kompatibilita. Ponúkané riešenie musí byť spätne kompatibilné so zariadeniami, ktoré sú v súčasnosti používané na prehliadanie webových stránok. Riešenie, ktoré by nebolo spätne kompatibilné, by nemohlo byť vývojármi vo veľkej miere implementované z dôvodu hrozby straty množstva užívateľov. Ďalším problémom, nad ktorým je potrebné sa zamyslieť je, aby riešenie bolo dostatočne univerzálne a v budúcnosti umožňovalo sémantiku rozširovať podľa požiadaviek, ktoré vzniknú.

5.1.1 Sémantika v Typo pomocou HTML5

WHATWG si pri návrhu HTML5 zvolila za riešenie sémantického webu zavedenie nových elementov. Kompletný zoznam týchto elementov sme si predstavili v kapitole 4.4. Pri návrhu nových elementov sa vychádzalo zo súčasných potrieb webových vývojárov. To znamená, že nové elementy nepokrývajú len oblasť sémantiky samotného obsahu webových stránok, ale pokrývajú taktiež aj samotnú štruktúru webových stránok.

Na webových stránkach môžeme identifikovať spoločné prvky akými sú napríklad hlavné menu, obsah, pätička stránky, jednotlivé články, komentáre atď. Typo nie je v tomto smere výnimkou a taktiež používa tieto spoločné prvky. Na výpise zdrojového kódu 31 môžeme vidieť akým spôsobom je označená hlavná navigácia v hlavnej šablóne systému Typo.

```
<ul id="navigation">
    <li><a href="http://localhost:3000" class="active">Home</a></li>
    <li><a href="/categories">Articles</a></li>
    <li><a href="http://localhost:3000/pages/about" >about</a></li>
</ul>
```

Výpis 31: Označenie hlavnej navigácie pomocou jazyka HTML4

V ukážkovom kóde môžeme vidieť akýsi pseudo kód (vytváraný pomocou atribútov *class* a *id*), ktorým sa vývojári snažia označiť, respektíve vniesť význam jednotlivým sekciám stránky. Problémom je, že každý autor si vytvára vlastný pseudo jazyk a len on samotný vie, aká časť stránky má aký význam. Keďže HTML5 zavádza nové elementy do jazyka HTML, my sa budeme snažiť vniesť sémantiku do tohto systému pomocou nových elementov. Jedným z nových elementov je aj element <nav>, ktorý sme si predstavili v kapitole 4.4.10. Označenie hlavnej navigácie pomocou tohto elementu môžeme vidieť na výpise kódu 32.

```
<nav id="navigation">
    <ul>
        <li><a href="http://localhost:3000">Home</a></li>
        <li><a href="/archives" rel="archives">Archives</a></li>
        <li><a href="http://localhost:3000/pages/about" rel="author">About</a></li>
    </ul>
</nav>
```

Výpis 32: Označenie hlavnej navigácie pomocou jazyka HTML5

Ako môžeme vidieť, pri návrhu nových elementov sa skutočne vychádzalo zo súčasných potrieb webových vývojárov a okrem elementov, ktoré prinášajú sémantiku do samotného obsahu, ako sú napríklad article, meter alebo mark, sú k dispozícii aj elementy, ktoré označujú samotnú štruktúru stránky ako sú footer, header, section alebo nav. Z tohto dôvodu bude jednou z úloh implementácie pretransformovať HTML 4 kód systému Typo do HTML5 kódu a vniesť význam samotnému obsahu systému ako i jeho ovládacím prvkom.

5.1.2 Sémantika pomocou Mikroformátov

Alternatívnym riešením, ako vniesť sémantiku do jazyka HTML sú Mikroformáty. Mikroformáty sú množina jednoduchých a otvorených dátových formátov, ktoré sú postavené nad existujúcimi a používanými štandardami. Pre pridanie štruktúry a významu textom na webových stránkach sa používajú metadáta a atribúty existujúcich (X)HTML elementov [13]. K tomuto účelu Mikroformáty využívajú atribúty *class*, *rel* a *rev*. Na výpise zdrojového kódu 33 môžeme vidieť, akým spôsobom by sme v HTML kóde označili blížiacu sa udalosť.

```
<span class="vevent">
    <span class="summary">The microformats.org site was launched</span>
    on <span class="dtstart">2011-06-20</span>
    at the Supernova Conference
    in <span class="location">San Francisco, CA, USA</span>.
</span>
```

Výpis 33: Označenie udalosti v HTML kóde pomocou mikroformátu hCalendar

Ako sme už spomínali, človek dokáže rozoznať takéto informácie od ostatných informácií napríklad tým, že je pri týchto informáciách uvedený dátum a čas. Avšak softvér, ktorý tieto informácie spracováva toto nedokáže, pokiaľ do HTML kódu nevnesieme sémantiku. Akonáhle do nášho kódu vnesieme sémantiku pomocou Mikroformátov, webový prehliadač, ktorý podporuje Mikroformáty, je schopný rozpoznať takúto udalosť a môže nám napríklad umožniť pridať udalosť priamo do nášho elektronického kalendára kliknutím na požadovaný dátum a čas.

5.1.3 HTML5 verzus Mikroformáty

Z úvodu do sémantiky pomocou HTML5 a Mikroformátov vyplýva, že každá z týchto technológii zvolila iný prístup. Zatiaľčo HTML5 sa snaží sémantiku rozšíriť pomocou nových elementov, Mikroformáty využívajú už existujúce elementy jazyka HTML a sémantiku rozširujú pomocou atribútov týchto elementov. Pri porovnávaní HTML5 a Mikroformátov sa treba zamyslieť aj nad výberom jednotlivých sémantických elementov. HTML5 sa snaží v porovnaní s Mikroformátmi pokryť omnoho väčšiu oblasť, kde sa zameriava nie len na samotný obsah webových stránok, ale taktiež na ich jednotlivé prvky ako sú hlavička, pätička, navigácie atď. Na druhej strane, Mikroformáty sa zameriavajú výlučne len na obsah webových stránok, kde ponúkajú sémantické elementy akými sú kalendár, vizitka, vzťahy medzi ľuďmi a dokumentami atď. V prípade Mikroformátov môžeme hovoriť o riešeniach, ktoré sa zameriavajú na špecifické situácie, zatiaľčo v prípade HTML5 môžeme hovoriť o všeobecnejších riešeniach sémantiky. Rozdiel v sémantike medzi HTML5 a mikroformátmi môžeme vidieť na výpise zdrojových kódov 34 a 35.

```
<div class="hreview">
    <span><span class="rating">5</span> out of 5 stars</span>
    <h4 class="summary">Crepes on Cole is awesome</h4>
    <span class="reviewer vcard">Reviewer: <span class="fn">Tantek</span> -
```

```
        <abbr class="dtreviewed" title="20050418T2300-0700">April 18, 2005</abbr></span>
    <div class="description item vcard"><p>
            <span class="fn org">Crepes on Cole</span> is one of the best little
            creperies in <span class="adr"><span class="locality">San Francisco</span></
                span>.
            Excellent food and service. </p></div>
    <p>Visit date: <span>April 2005</span></p>
    <p>Food eaten: <span>Florentine crepe</span></p>
</div>
```

Výpis 34: Recenzia vytvorená pomocou Mikroformátov

```
<article class="review">
    <header>
        <h1>Crepes on Cole is awesome</h1>
        <meter min="1" max="5" value="5">5 out of 5 stars</meter>
        <span class="reviewer">Reviewer: Tantek</span> -
        <time datetime="2005-04-18T23:00-07:00" pubdate>April 18, 2005</time>
    </header>
    <p><span>Crepes on Cole</span> is one of the best little creperies in <span>San
        Francisco</span>.</p>
    <p>Visit date: <time datetime="2005-04-18">April 2005</time></p>
    <p>Food eaten: <span>Florentine crepe</span></p>
</article>
```

Výpis 35: Recenzia vytvorená pomocou HTML5

V úvode kapitoly 5.1 sme si položili dve otázky, ktoré kládli dôraz na univerzálnosť a rozšíriteľnosť.

5.1.4 Spätná kompatibilita s aktuálne používanými zariadeniami

Ako sa uvádza v dokumente HTML5 differences from HTML4 [14], HTML5 je navrhnuté spôsobom, ktorý je spätne kompatibilný so spôsobom, akým softvér spracováva súčasný webový obsah. To znamená, že prehliadače, ktoré nepodporujú nové elementy, spracovávajú tieto nové elementy štandardným spôsobom ako elementy jazyka HTML 4. Výsledkom takéhoto spracovávania je korektný DOM strom a správne zobrazená webová stránka. Avšak problém nastáva v prehliadačoch Internet Explorer verzie 6, 7 a 8, kde prehliadač síce spracuje kód správne, ale výsledne vytvorený DOM strom nie je korektný a výsledkom je nesprávne zobrazená webová stránka. Viac sa tomuto problému venuje Lachlan Hung [15], kde ponúka aj jednoduché riešenie pomocou javascriptu a výsledkom je korektný DOM strom a správne fungujúca webová stránka. Pri Mikroformátoch je situácia veľmi jasná, kde už z povahy Mikroformátov vyplýva, že spätná kompatibilita je zaručená a to z dôvodu, že Mikroformáty sú postavené na existujúcich technológiách a predstavujú akýsi doplnok jazyka HTML. Odpoveďou na položenú otázku je, že HTML5 a aj Mikroformáty sú spätne kompatibilné so súčasnými zariadeniami.

5.1.5 Možnosť budúceho rozširovania

HTML5 a Mikroformáty vyznávajú dva rôzne prístupy. HTML5 rieši sémantiku cestou zavedenia nových elementov. Z tohto pohľadu je jasné, že možnosť jednoduchého rozšírenia jazyka v budúcnosti nie je možná a každé zavedenie nových elementov bude musieť viesť aj k zmene samotnej gramatiky jazyka HTML. Následne budú musieť tieto zmeny implementovať prehliadače, aby boli schopné nové elementy korektne spracovať. Možnosť budúceho rozširovania z pohľadu Mikroformátov je omnoho jednoduchšia, keďže pri zavedení nových rozšírení nie je potrebné zavádzať žiadne nové elementy. V prípade nových mikroformátov nám stačí zaviesť nové hodnoty atribútov elementom v jazyku HTML. Situácia je obdobne jednoduchšia aj pre prehliadače, keďže sa nemení gramatika jazyka HTML.

Myslím si, že HTML5 sa v oblasti sémantiky vydal správnym smerom. Vynorilo sa však aj množstvo otázok, prečo boli vytvorené elementy, ktoré pokrývajú len určitú oblasť a neboli vytvorené aj iné elementy, ktoré by pokrývali aj iné oblasti. Avšak osobne si myslím, že sémantika v kontexte nových elementov a HTML je postačujúca a cieľom HTML5 nebolo vniesť sémantiku do jazyka spôsobom, ktorý by dokázal pokryť oblasť všetkých dostupných informácií. Práve k tomuto účelu slúžia rôzne iné prostriedky, akými sú RDF alebo OWL, a práve tieto prostriedky môžu v budúcnosti zohrať významnú úlohu ako doplnok jazyka HTML5. Pre bližšie štúdium sémantiky odporúčam dokument Sémantický web [32].

5.2 Offline mód

Internet a s ním aj webové stránky v súčasnosti prenikli do mnohých oblastí nášho života a pomocou webových stránok dokážeme vykonávať mnohé činnosti, na ktoré boli predtým potrebné špecializované desktopové aplikácie. Veľkou výhodou, ale zároveň však aj nedostatkom týchto webových stránok je ich závislosť práve na sieti internet. Určite nám princíp webových stránok v mnohých oblastiach uľahčuje prácu. Webové stránky sú dostupné vždy a všade tam, kde máme prístup k internetu a internetovému prehliadaču, avšak z určitého pohľadu je práve závislosť na pripojení, a teda požiadavka byť neustále online, slabým miestom celého mechanizmu. K vyriešeniu tohoto problému ponúka HTML5 offline mód, ktorý umožní fungovanie webových stránok bez internetového pripojenia, a ktorý sme si predstavili v kapitole 4.7.

5.2.1 Offline mód v Typo pomocou HTML5

Pomocou systémov pre správu obsahu dokážeme zobraziť množstvo informácií, typicky sú to rôzne články, stránky, komentáre atď. Na tomto mieste sa vynára úvaha, akým spôsobom zabezpečiť, aby bolo možné pracovať s týmito informáciami aj v prípade dočasnej straty pripojenia. V praxi si to môžeme predstaviť tak, že užívateľ alebo internetový prehliadač si stiahne potrebné dáta a uloží ich do špeciálnej pamäte, a užívateľ následne môže so stránkou plnohodnotne pracovať aj v prípade, že momentálne nie je pripojený. Úlohou implementácie bude preto rozšíriť systém Typo o možnosť práce

s týmto systémom aj počas nedostupnosti pripojenia. Implementovanie offline módu do systému Typo môžeme rozdeliť na dve časti. Prvou časťou je uloženie potrebných súborov do internej pamäte prehliadača. K tomuto účelu využijeme offline API a offline mód, ktorý ponúka HTML5. Druhou časťou je umožniť užívateľovi plnohodnotne pracovať so stránkou, to znamená umožniť mu vkladať komentáre a čítať rôzne články. K tomuto účelu využije Web SQL Database, ktorú ponúka HTML5.

5.2.2 Offline mód pomocou Google Gears

Google Gears predstavuje technológiu, ktorá umožňuje fungovanie webových stránok aj bez internetového pripojenia. Ako už názov napovedá, technológiu vyvinula firma Google, a technológia je distribuovaná ako rozšírenie pre jednotlivé internetové prehliadače. Celá technológia sa skladá z niekoľkých modulov, ktorými sú LocalServer, Database a WorkerPool.

LocalServer - Úlohou modulu LocalServer a jeho príslušného API je umožniť webovým stránkam pracovať v režime offline. To znamená, že modul predstavuje špecializovanú cache pamäť, ktorá ukladá URL adresy a ich obsah. Požiadavky na tieto URL adresy sú spracovávané a servírované z lokálneho disku. Základom celého mechanizmu je ako v prípade HTML5 manifest súbor, ktorý obsahuje zoznam URL adries na požadované súbory, ktoré majú byť uložené do cache pamäte.

Database - Aplikácie, ktoré nie sú len obyčajné statické stránky, ukladajú svoje dáta typicky na serveri. Aby mohli byť tieto aplikácie užitočné aj v offline móde, tieto dáta musia byť dostupné aj lokálne. K tomuto účelu Google Gears poskytuje modul Database, ktorý poskytuje relačnú databázu pre ukladanie dát. Celý mechanizmus je postavený na relačnej databáze SQLite [2].

WorkerPool - Niektoré operácie (napr. synchronizácia veľkého množstva dát) si vyžadujú určitý čas a prostriedky a v mnohých prípadoch môžu negatívne ovplyvňovať výkon a schopnosť rýchlej odozvy internetového prehliadača. Modul WorkerPool dokáže takéto operácie spúšťať na pozadí a umožňuje internetovému prehliadaču schopnosť rýchlej odozvy na užívateľove požiadavky. WorkerPool môžeme v tomto kontexte chápať ako mechanizmus javascriptových vlákien.

5.2.3 HTML5 verzus Google Gears

Ako sme si mohli všimnúť, celý mechanizmus offline módu v HTML5 vychádza práve z technológie Google Gears a tieto technológie sú svojím spôsobom poňatia problému takmer identické. Nie je sa ani čomu čudovať, keďže hlavný autor HTML5 Ian Hickson je zamestnancom firmy Google. Hlavným rozdielom je spôsob fungovania týchto mechanizmov, kde v prípade Google Gears potrebujeme špecializované rozšírenie dostupné pre konkrétny internetový prehliadač na danej platforme, zatiaľčo v prípade HTML5 nie sú potrebné žiadne špeciálne rozšírenia.

Myslím si, že rozhodnutie vziať dobre fungujúcu a používanú technológiu, a začleniť ju do štandardu HTML5 bolo správnym krokom, ktorý potvrdzuje aj fakt, že firma Google sa rozhodla ukončiť vývoj Google Gears na úkor HTML5. Podobnosť technológií

HTML5 a Google Gears môžeme vidieť na výpise zdrojových kódov 36 a 37.

```
CACHE MANIFEST
#version: 1

site.html
gears_init.js
```

Výpis 36: Syntax manifest súboru v HTML5

```
{
  "betaManifestVersion": 1,
  "version": " site_version_string ",
  "entries": [
    { "url": "site.html" },
    { "url": "gears_init.js" }
  ]
}
```

Výpis 37: Syntax manifest súboru v Google Gears

5.2.4 Offline aplikácie pomocou Adobe AIR

Adobe AIR predstavuje technológiu, ktorá umožňuje vývojárom vytvárať RICH internetové aplikácie použitím osvedčených technológií, akými sú HTML, Javascript, Actionscript alebo Flash. Jednou z vlastností takýchto aplikácií je, že dokážu pracovať aj mimo internetového prehliadača a na rôznych operačných systémoch. Cieľom celej technológie je priblížiť internetové aplikácie tým desktopovým. Jednou z hlavných vlastností aplikácií postavených na Adobe AIR je aj možnosť práce offline, to znamená, že aplikácie sú schopné plnohodnotne pracovať aj v prípade nedostupnosti internetového pripojenia. Adobe AIR aplikácie môžu svoje dáta v prípade offline módu ukladať do relačnej databázy SQLite [2].

5.2.5 HTML5 verzus Adobe AIR

Hlavným rozdielom medzi HTML5 a Adobe AIR je v spôsobe nasadenia aplikácií. HTML5 aplikácie využívajú k svojmu behu primárne internetový prehliadač, zatiaľčo Adobe AIR aplikácie využívajú vlastné behové prostredie, ktoré je nezávislé na internetovom prehliadači. Každý z týchto prístupov má svoje výhody aj nevýhody. HTML5 aplikácie, ktoré bežia v internetovom prehliadači nevyžadujú žiadnu inštaláciu, zatiaľčo Adobe AIR aplikácie vyžadujú vytvorenie inštalačného balíčku, jeho digitálne podpísanie a následnú inštaláciu na užívateľov lokálny súborový systém. Na druhej strane z tohto prístupu plynú aj výhody pre Adobe AIR aplikácie a to je napríklad prístup k lokálnym prostriedkom počítača, zatiaľčo možnosti webových stránok sú v tomto smere veľmi obmedzené. Ďalším rozdielom sú aj použité programovacie jazyky. V prípade HTML5 máme k dispozícii HTML v kombinácii s Javascriptom, v prípade Adobe AIR môžeme k vytvoreniu aplikácií použiť okrem HTML a Javascriptu, Flash alebo Flex.

5.3 SVG

SVG predstavuje jazyk založený na XML, ktorý slúži pre popis 2D vektorovej grafiky. Vektorová grafika je grafika, ktorá je založená na matematických rovniciach a grafické objekty zobrazuje pomocou geometrických prvkov akými sú body, čiary, krivky a iné. Z vlastnosti, že geometrické prvky sú vyjadrené pomocou matematických rovníc, plynú mnohé výhody. Jednou z týchto výhod je, že grafika môže byť ľubovoľne škálovateľná bez straty kvality zobrazovaného objektu. Z tohto dôvodu je SVG vhodným kandidátom, prostredníctvom ktorého dokážeme vytvoriť taký grafický výstup, ktorý by bol interpretovaný nezmenene na rôznych zariadeniach s rôznymi rozlíšeniami obrazovky.

5.3.1 SVG v Typo pomocou HTML5

V súčasnosti sa čoraz väčšej popularite tešia rôzne inteligentné telefóny alebo internetové tablety, ktoré v mnohých oblastiach konkurujú bežným počítačom. Jednou z takýchto oblastí je aj oblasť prehliadania Internetu. Takéto zariadenia dokážu plnohodnotne zobraziť rôzne webové stránky pomocou svojich internetových prehliadačov, ktoré sa v mnohých oblastiach vyrovnávajú prehliadačom, ktoré bežne používame na stolných počítačoch. Komplikáciou z pohľadu webových stránok je, že každé z takýchto zariadení používa rôzne rozlíšenie obrazovky. Keďže webové stránky predstavujú médium, ktoré je závislé na rozlíšení, naskytuje sa nám tu otázka, akým spôsobom zabezpečiť, aby navrhovaná stránka a jej grafické prvky boli správne zobrazené na všetkých resp. na väčšine takýchto zariadení. V tomto kontexte môžeme za správne zobrazenie grafických prvkov považovať zobrazenie, ktoré je nezávislé na rozlíšení a v prípade ľubovoľného zmenšenia alebo zväčšenia grafických prvkov nedochádza k strate zobrazovaných informácií. Odpoveď na položenú otázku nám dáva SVG. HTML5 nám umožňuje použiť SVG v rámci HTML kódu ako bežné elementy jazyka HTML, a preto úlohou implementácie bude vytvoriť takú verziu systému Typo, aby grafické prvky boli zobrazené správne bez straty informácií pri ľubovoľnom rozlíšení. Takéto grafické prvky budú reprezentované SVG grafikou. V tomto kontexte môžeme úlohu chápať ako vytvorenie verzie systému určenej pre mobilné zariadenia.

5.3.2 SVG v HTML5 verzus iné riešenia

V súčasnosti môžeme k vytvoreniu webových stránok pre mobilné zariadenia použiť niekoľko postupov. Jednou z možností je napríklad použitie atribútu *handheld* pri vkladaní css súborov do našich webových stránok, ktorým určujeme, že daný css súbor sa má aplikovať len na prenosné zariadenia, typicky s malou obrazovkou a limitovanou rýchlosťou pripojenia. Ďalšou možnosťou je vytvorenie špeciálnej verzie webových stránok, ktorá je dostupná z vyhradenej URL a slúži buď ako špeciálna verzia primárne určená pre mobilné zariadenia alebo môže slúžiť ako špeciálna verzia vyhradená pre konkrétne mobilné zariadenie. Ďalšou možnosťou je identifikácia typov jednotlivých prehliadačov a na základe týchto typov môžeme zobrazovať špeciálne verzie stránok daným zariadeniam. V súčasnosti sa môžeme taktiež stretnúť s možnosťou tvorby WAP stránok, ktoré sú

napísané pomocou jazyka WML. Vo väčšine spomínaných postupov je výsledná stránka graficky veľmi jednoduchá a to z dôvodu, aby zabezpečila správne zobrazenie grafických prvkov a informácií na čo najrôznejších mobilných zariadeniach. Naproti tomu máme možnosť vytvoriť webovú stránku pre mobilné zariadenie v kombinácii HTML a SVG, ktorá nám umožňuje vytvárať bohaté grafické prvky, ktorých výsledná zobrazovaná kvalita je nezávislá od rozlíšenia obrazovky zariadenia, ktoré ich zobrazuje. Na druhej strane by bolo veľmi obtiažne všetky informácie a prvky webových stránok zobrazovať prostredníctvom SVG, preto ako najvhodnejšie riešenie je použitie SVG ako vhodného doplnku spomínaných postupov pre tvorbu mobilných verzií webových stránok.

5.4 HTML5 a javascript

HTML5 prinieslo aj niekoľko noviniek, ktoré významným spôsobom uľahčujú prácu s webovými stránkami a vnášajú zvyklosti, ktoré poznáme z bežných desktopových aplikácií. Jedná sa napríklad o už spomínané vlastnosti ako sú natívne Drag and Drop API, editovateľný obsah pomocou atribútu *contenteditable* alebo nové input prvky, ktoré umožňujú vytvoriť napríklad v prípade zadávania dátumu plnohodnotný kalendár. Pri porovnaní HTML5 s javascriptovými knižnicami sa obmedzíme na knižnicu jQuery [18], ktorá v súčasnosti predstavuje najpoužívanejšiu a najkomplexnejšiu dostupnú knižnicu.

5.4.1 Drag and Drop v HTML5

Drag-and-drop mechanizmus sme si bližšie predstavili v kapitole 4.8 a povedali sme si, že nastavením hodnoty atribútu *dragable* na true umožníme, aby mohol byť daný element ľubovoľne presúvaný v rámci stránky. Následne môžeme pracovať s takýmto elementom na základe rôznych udalostí ako sme si ukázali na výpise zdrojového kódu 30.

5.4.2 Drag and Drop v jQuery

Mechanizmus drag-and-drop ponúka knižnica jQuery prostredníctvom rôznych rozšírení. Riešenie ktoré ponúka knižnica jQuery sa odlišuje najmä použitou syntaxou. Princíp ostáva však nezmenený. Najprv musíme určiť, ktorý element má mať vlastnosť draggable a následne pomocou dostupných udalostí môžeme pracovať s takýmto elementom podľa stavu, v ktorom sa práve nachádza.

5.4.3 Drag and Drop v HTML5 verzus jQuery

Ako záver môžeme vyvodiť, že HTML5 štandardizovalo spôsob, ktorý môžeme používať aj v súčasnosti, avšak boli k tomu potrebné rôzne javascriptové knižnice. Porovnanie Drag and Drop implementácie v HTML5 a v jQuery môžeme vidieť na výpisoch zdrojových kódov 38 a 39.

```
<div draggable="true">drag and drop</div>
```

Výpis 38: Ukážka použitia Drag and Drop v HTML5

```
<script>
    $().ready(function() {
        $('#draggable').Drags({});
    });
</script>
<div id="draggable">drag and drop</div>
```

Výpis 39: Ukážka použitia Drag and Drop v jQuery

5.4.4 Editovateľné elementy v HTML5

HTML5 prináša do jazyka ďalšiu vlastnosť, na ktorej realizáciu bolo doteraz nutné použitie javascriptovej knižnice. Touto vlastnosťou sú editovateľné elementy, ktorú sme si predstavili v kapitole 4.9.

5.4.5 Editovateľné elementy v jQuery

Editovateľné elementy ponúka knižnica jQuery pomocou rôznych rozšírení. Ponúkané riešenie sa v porovnaní s HTML5 odlišuje predovšetkým použitou syntaxou.

5.4.6 Editovateľné elementy v HTML5 verzus jQuery

Ako záver môžeme opäť vyvodiť, že HTML5 len štandardizovalo spôsob, ktorým sme mohli pracovať aj doteraz, avšak boli k tomu potrebné rôzne javascriptové knižnice. Porovnanie implementácie editovateľných elementov môžeme vidieť na výpisoch zdrojových kódov 40 a 41.

```
<section contenteditable="true" id="editable">
    <h1>Title</h1>
    <p>Text</p>
</section>
```

Výpis 40: Ukážka editovateľného elementu v HTML5

```
<script>
    var editable = new myEditable('#editable',{});
</script>

<section id="editable">
    <h1>Title</h1>
    <p>Text</p>
</section>
```

Výpis 41: Ukážka editovateľného elementu v jQuery

6 Implementácia offline módu

Cieľom implementácie offline módu pre systém Typo je vytvoriť takú verziu systému, s ktorou bude môcť užívateľ pracovať aj v prípade, že v danom čase nie je pripojený.

6.1 Špecifikácia zadania

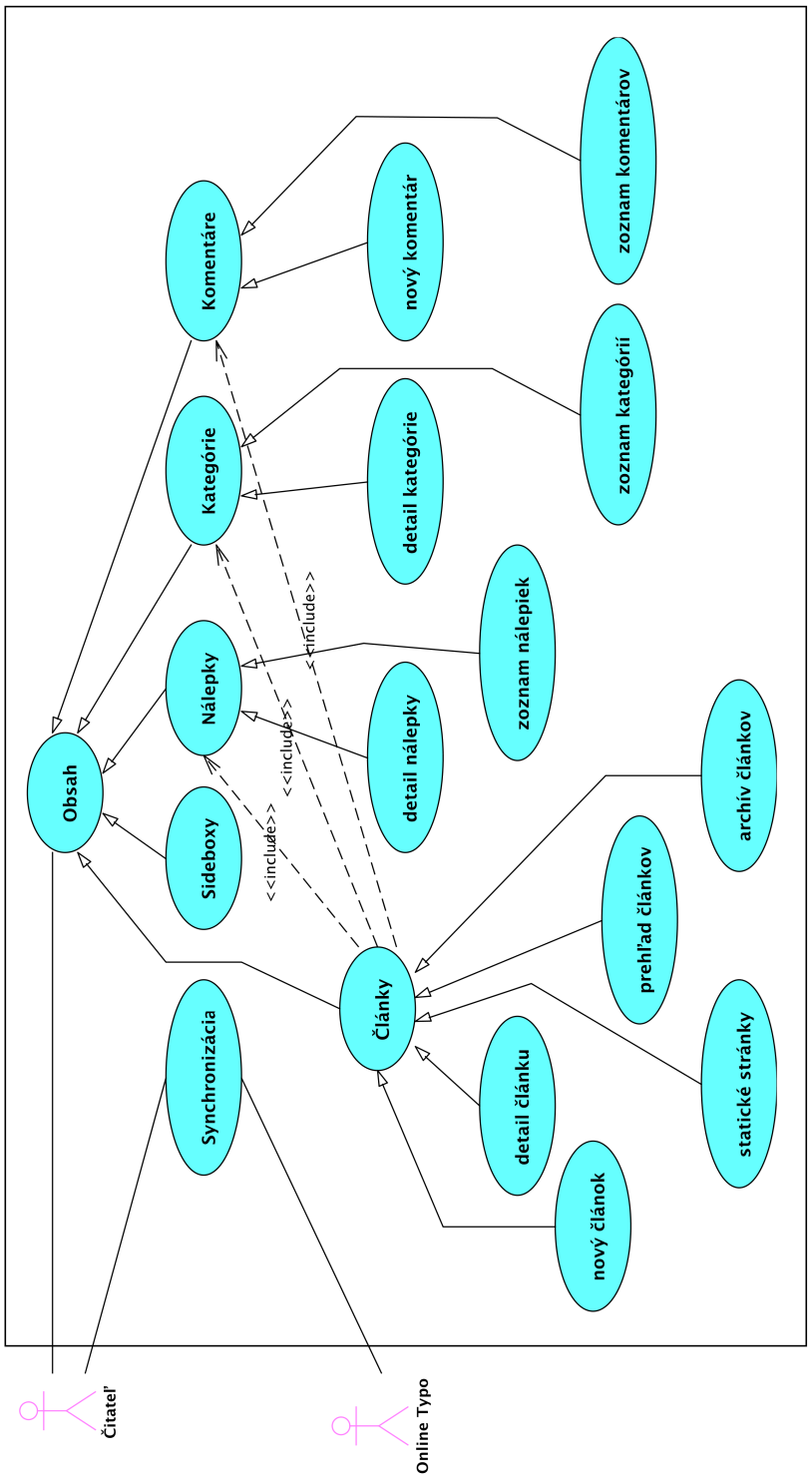
V offline móde bude mať užívateľ možnosť prezerať vybrané články, kategórie článkov alebo nálepky jednotlivých článkov. Užívateľovi budú taktiež prístupné sideboxy, ktoré zobrazujú dôležité informácie akými sú zoznam kategórií, zoznam nálepiek alebo zoznam statických stránok systému. Ďalšou vlastnosťou systému je možnosť pridávať komentáre k jednotlivým článkom a taktiež pridávať nové články. Jednou z najdôležitejších a kľúčových vlastností offline verzie systému je možnosť synchronizácie dát s online verziou systému. Z tohto dôvodu bude mať užívateľ k dispozícii nástroje, ktoré mu umožnia synchronizovať takéto dáta.

6.2 Špecifikácia požiadaviek

Cieľom špecifikácie požiadaviek je popísať, ako má softvérový systém pracovať prostredníctvom špecifikácie jeho funkcionality [25]. K tomuto účelu využijeme diagramy prípadov použitia (use case). Tieto diagramy definujú vzťahy medzi prípadmi použitia systému a aktérmi vystupujúcimi vo vnútri systému. Prípady použitia špecifikujú vzory chovania realizované softvérovým systémom. Každý prípad použitia možno chápať ako postupnosť vzájomne nadväzujúcich transakcií vykonaných v dialógu medzi aktérom a vlastným softvérovým systémom. Ako aktérov potom definujeme užívateľov alebo iné systémy, ktoré budú vstupovať do interakcie s vyvíjaným systémom [26]. V prvom kroku sa zameriame na systém ako celok, kde ukážeme pomocou use case diagramu najdôležitejšie prípady použitia, ktoré tvoria základ celého systému. V ďalšom kroku si vybrané prípady použitia bližšie špecifikujeme pomocou detailnejších use case diagramov a v poslednom kroku si na vybraných diagramoch ukážeme špecifikáciu ich chovania. V rozsahu tejto práce nie je možné detailne popísať celú funkcionalitu systému, preto sa v špecifikácii požiadaviek zameriame len na vybrané najdôležitejšie prípady použitia, ktoré identifikujeme v úvodnom use case diagrame.

6.2.1 Systém

Diagram na obrázku 1 popisuje užívateľov alebo systémy, ktorí vstupujú do interakcie so systémom. Čitateľ má prístup k obsahu systému, ktorý predstavujú články, statické stránky, komentáre, kategórie, nálepky a sideboxy. Čitateľ môže zobrazovať prehľad všetkých článkov ako i jednotlivé články a statické stránky a taktiež môže pridávať nové články. Čitateľ má taktiež k dispozícii zobrazenie komentárov k článkom a pridávanie nových komentárov. Čitateľ má prístup k zobrazeniu prehľadu kategórií a nálepiek ako i podrobný zoznam článkov, ktoré sa nachádzajú v kategóriách respektíve nálepkách. Čitateľ môže vykonávať synchronizáciu dát s online verziou systému.



Obr. 1: Use case - Systém

6.2.1.1 Zobrazenie článku

Hlavným zámerom procesu je umožniť užívateľovi zobraziť článok spolu so súvisiacimi informáciami, akými sú nálepky a komentáre k článku.

Účastníci: Čitateľ
Vstupná podmienka:

- V systéme sa nachádza minimálne jeden článok

Záruky úspechu: Čitateľovi sa zobrazí článok
Hlavný scenár

1. Systém zobrazí zoznam článkov
2. Čitateľ zvolí článok pre zobrazenie
3. Systém zobrazí článok

Rozšírenia

3.a Zobrazenie článku bolo neúspešné

3.a.1 Systém informuje čitateľa o neúspešnom zobrazení článku

6.2.1.2 Pridanie komentára

Hlavným zámerom procesu pridania komentára je umožniť užívateľovi, aby ku konkrétnemu článku mohol pridať komentár.

Účastníci: Čitateľ
Vstupná podmienka:

- V systéme sa nachádza minimálne jeden článok
- Čitateľ sa nachádza v sekcii konkrétneho článku
- K článku je povolené vkladanie nových komentárov

Záruky úspechu: Čitateľ vloží nový komentár
Hlavný scenár

1. Čitateľ zvolí pridanie nového komentára
2. Čitateľ vloží údaje a obsah komentára
3. Systém overí správnosť vkladaných údajov
4. Systém uloží komentár
5. Systém zobrazí vložený komentár

Rozšírenia

3.a Čitateľ vložil nesprávne údaje

3.a.1 Systém informuje čitateľa o neúspešnom uložení komentára a požiada o nové údaje

3.a.2 Čitateľ vloží nové údaje

3.b Komentáre k článku nie sú povolené

3.b.1 Systém informuje čitateľa o nemožnosti vkladať komentáre k článku

6.2.1.3 Zobrazenie archívu článkov Hlavným zámerom procesu je umožniť užívateľovi prezerať archív článkov zotriedený podľa jednotlivých mesiacov v roku.

Účastníci: Čitateľ
Vstupná podmienka:

- V systéme sa nachádza minimálne jeden článok

Záruky úspechu: Čitateľovi sa zobrazí archív článkov
Hlavný scenár

1. Systém zobrazí archív článkov
2. Čitateľ zvolí mesiac a rok
3. Systém zobrazí články podľa zvoleného mesiaca a roku
4. Čitateľ zvolí článok pre zobrazenie *(viď. 6.2.1.1 UC Zobrazenie článku)*

Rozšírenia

3.a Pre zvolený mesiac a rok neexistujú žiadne články

3.a.1 Systém informuje čitateľa o neúspešnom zobrazení zoznamu článkov

6.2.1.4 Synchronizácia údajov Hlavným zámerom procesu je umožniť čitateľovi synchronizovať dáta s online verziou systému.

Účastníci: Čitateľ
Vstupná podmienka:

- Čitateľ je pripojený k internetu
- Online verzia systému umožňuje synchronizáciu údajov

Záruky úspechu: Čitateľ má aktuálne dáta
Hlavný scenár

1. Čitateľ zvolí spustenie synchronizácie

2. Systém zobrazí čitateľovi správu s otázkou o začatí synchronizácie
3. Čitateľ potvrdí začatie synchronizácie
4. Systém vykoná synchronizáciu
5. Systém informuje čitateľa o výsledku synchronizácie

Rozšírenia

3.a Čitateľ nepotvrdí začatie synchronizácie

3.a.1 Systém informuje čitateľa o ukončení procesu

4.a Synchronizácia neúspešná

4.a.1 Systém informuje čitateľa o neúspešnej synchronizácii

6.3 Špecifikácia chovania

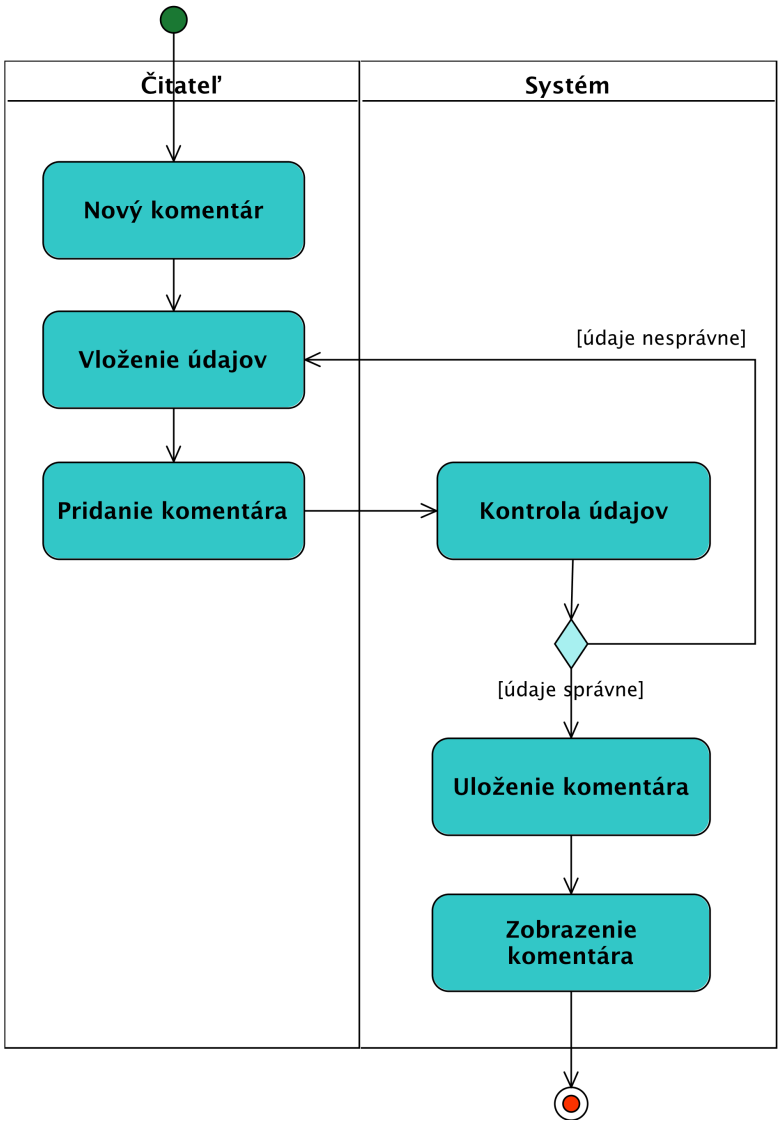
V druhej fáze špecifikácie požiadaviek budeme analyzovať prípady použitia, ktoré sme vytvorili v predchádzajúcom kroku. K tomuto účelu použijeme aktivitné diagramy. Pomocou týchto diagramov budeme popisovať jednotlivé procesy pomocou ich stavov reprezentovaných vykonávaním aktivít a pomocou prechodov medzi týmito stavmi spôsobených ukončením týchto aktivít. Ďalej definujeme, ktorý objekt zodpovedá za danú aktivitu, prípadne aké objekty sú aktivitami vytvárané, spotrebovávané alebo modifikované. Zameriame sa pri tom na dva procesy Pridanie komentára a Synchronizáciu údajov.

6.3.1 Pridanie komentára

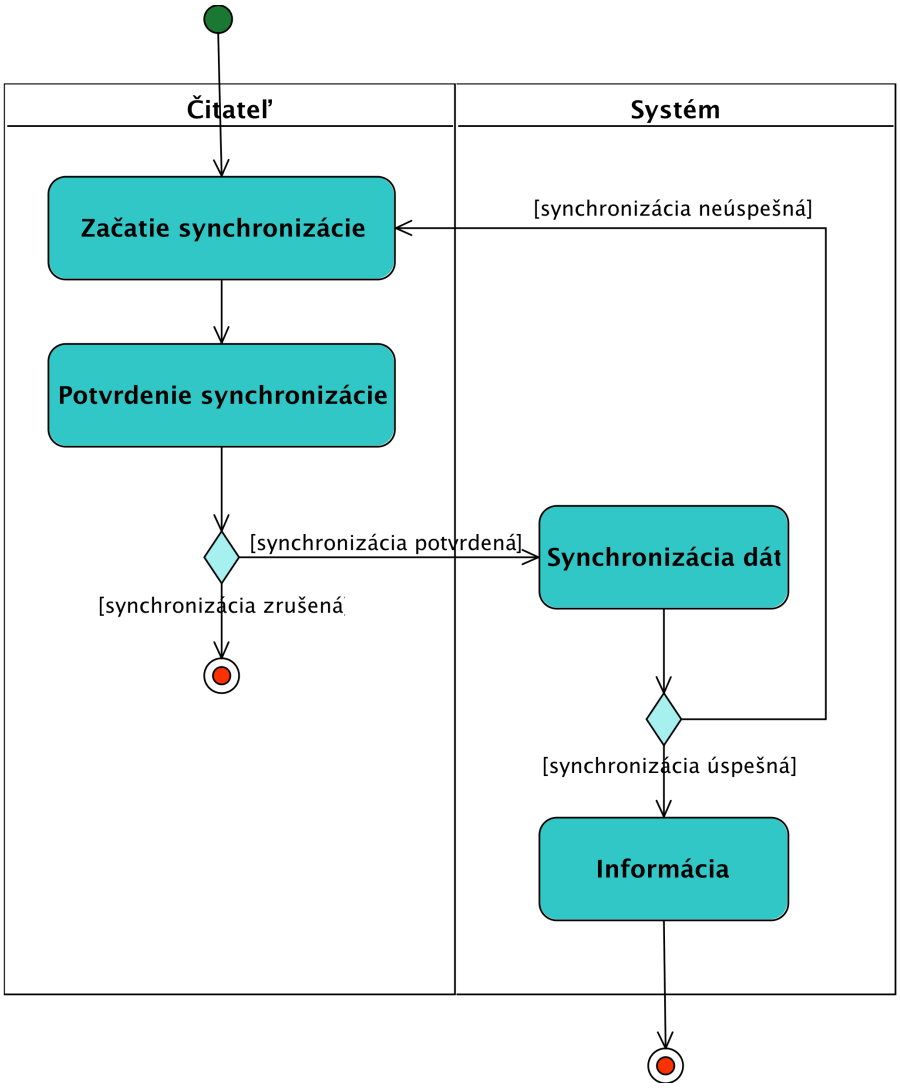
Na obrázku 2 môžeme vidieť diagram pre tento proces. Proces začína aktivitou pridaním nového komentára. Nasleduje vloženie samotného obsahu komentára. Po vložení údajov komentára nasleduje akcia pridanie komentára, v ktorej čitateľ odošle dáta systému. Systém následne skontroluje správnosť odoslaných dát. Ak dáta nie sú správne, užívateľ môže vložiť nové alebo upraviť existujúce údaje. Ak sú dáta správne, systém ich uloží a následne zobrazí pridaný komentár.

6.3.2 Synchronizácia údajov

Na obrázku 3 môžeme vidieť diagram pre tento proces. Proces začína aktivitou začatie synchronizácie, kde čitateľ spúšťa proces. Následne musí čitateľ potvrdiť spustenie procesu. Ak sa čitateľ rozhodne nepotvrdiť začatie synchronizácie, proces končí. V prípade potvrdenia synchronizácie systém vykoná synchronizáciu údajov. Ak synchronizácia skončila neúspechom, užívateľ má možnosť spustiť celý proces odznova. V prípade úspechu systém informuje užívateľa o výsledku synchronizácie.



Obr. 2: Aktivitný diagram - Pridanie komentára



Obr. 3: Aktivitný diagram - Synchronizácia údajov

6.4 Analýza

Cieľom analýzy a návrhu softvérového produktu je ukázať, akým spôsobom bude produkt realizovaný v implementačnej fáze [25].

Model analýzy skúma špecifikované požiadavky z pohľadu objektov, ktoré sa vyskytujú v problémovej doméne [25]. V predchádzajúcej časti sme si popísali základné prípady použitia, ktoré sa vyskytujú v systéme. V tejto časti budeme tieto prípady použitia skúmať z pohľadu objektov.

6.4.1 Dynamická štruktúra

Vo vzťahu k definovaným prípadom použitia je nutné definovať také interakcie medzi objektami, ktoré povedú k splneniu ich funkcionality a účelu, ku ktorému boli navrhnuté [25]. K zachyteniu vzájomných interakcií využijeme tzv. sekvenčné diagramy. V rozsahu tejto práce nie je možné popísať všetky prípady použitia, pretože prácnosť s tým spojená by prevýšila svoj účel. Z tohto dôvodu si vytvoríme sekvenčný diagram architektúry MVC. Architektúra MVC definuje vzťahy medzi jednotlivými časťami aplikácie a tým ovplyvňuje aj volania medzi jednotlivými objektami týchto častí. Aby sme si bližšie vysvetlili interný mechanizmus volania objektov v rámci aplikácie, vytvoríme si preto sekvenčný diagram, ktorý zobrazuje volania medzi jednotlivými časťami aplikácie. Výsledný diagram môžeme chápať ako abstrakciu interných volaní medzi jednotlivými časťami aplikácie.

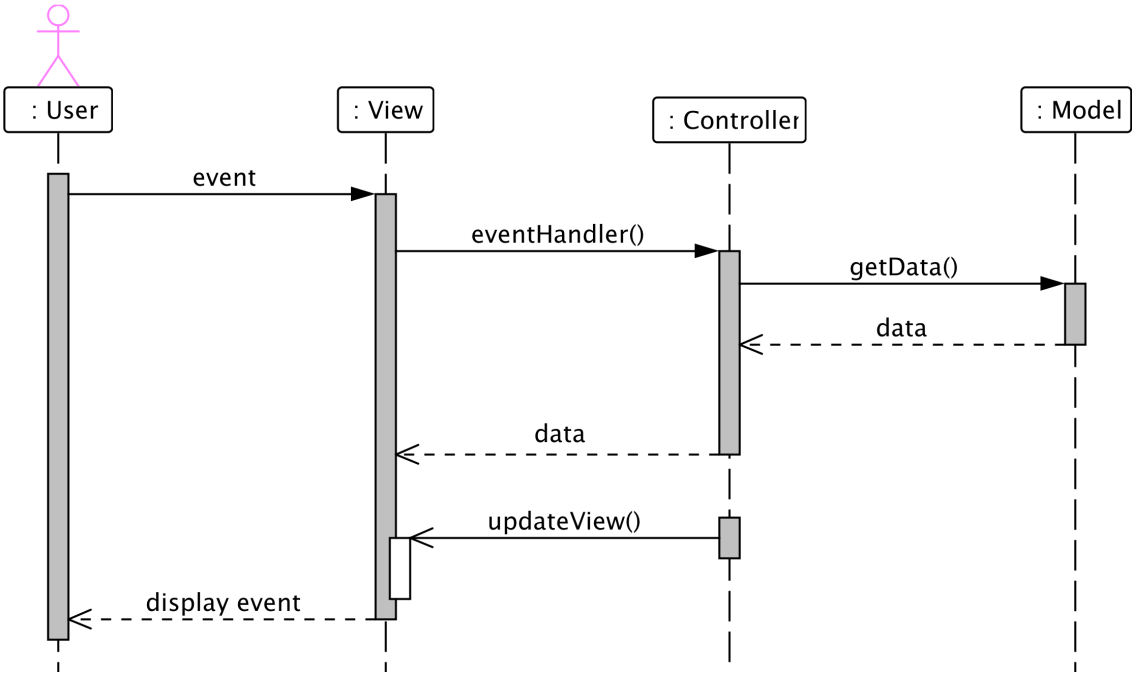
6.4.1.1 Sekvenčný diagram MVC Na obrázku 4 môžeme vidieť sekvenčný diagram, ktorý zobrazuje jeden z možných scenárov zachytenia a spracovania udalosti v rámci aplikácie. Scenár je realizovaný interakciou medzi užívateľom a objektami view, controller a model. Proces zahajuje užívateľ spustením udalosti *event* nad objektom view. Objekt view následne posiela správu *eventHandler()* objektu controller. Reakciou na takúto správu je zaslanie správy *getData()* objektu model pre získanie dát. Objekt model posiela ako návratovú správu požadované dáta. Objekt controller predáva získané dáta objektu view pomocou návratovej správy a posiela správu *updateView()* pre zobrazenie požadovaných dát.

6.4.2 Statická štruktúra

V tejto fáze analýzy systému upresníme jeho statický popis, ktorý sme vytvorili v rámci špecifikácie požiadaviek. K tomuto účelu použijeme výsledný diagram tried, ktorý môžeme vidieť na obrázku 5.

6.4.3 Dátový model

Cieľom dátového modelovania je navrhnúť kvalitnú dátovú štruktúru pre konkrétnu aplikáciu a databázový systém, ktorý bude táto aplikácia využívať k uloženiu dát. My sa



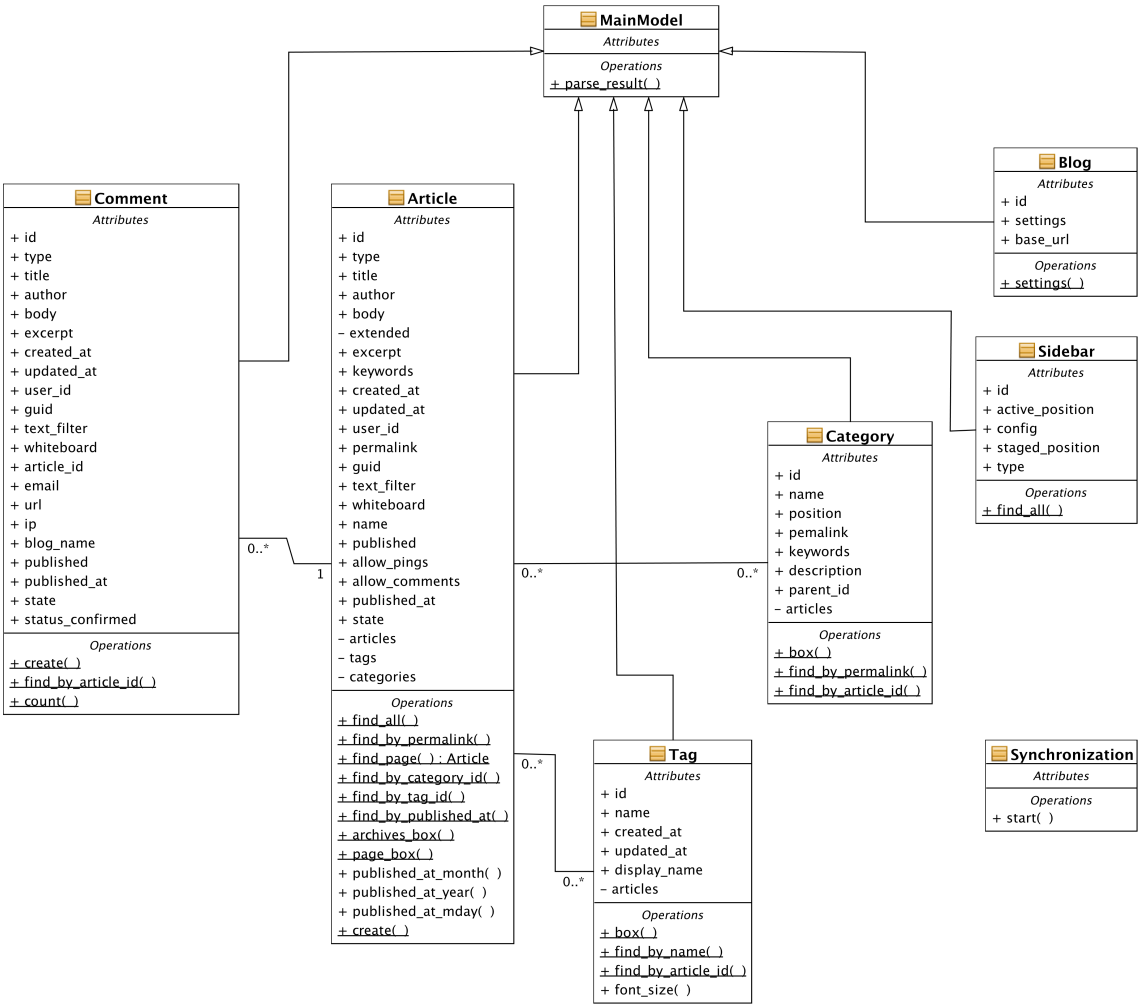
Obr. 4: Sekvenčný diagram - MVC

zameriame na vytvorenie konceptuálneho dátového modelu. Konceptuálny dátový model umožňuje vytvoriť popis dát v databáze, t.j. konceptuálnu schému, nezávisle na fyzickom uložení dát v databáze. Základnou komponentou konceptuálneho modelu je ER diagram. My však pre konceptuálne modelovanie využijeme možnosti jazyka UML. Jazyk UML nám poskytuje rovnaké možnosti modelovania ako ER diagram, ale hlavným dôvodom pre uprednostnenie jazyka UML pred ER diagramom je fakt, že pre analýzu aplikácie ako i analýzu databázy použijeme rovnaký modelovací jazyk. Výsledný dátový model môžeme vidieť na obrázku 6. Z dôvodu prehľadnosti takéhoto diagramu nie sú uvedené jednotlivé atribúty tabuliek.

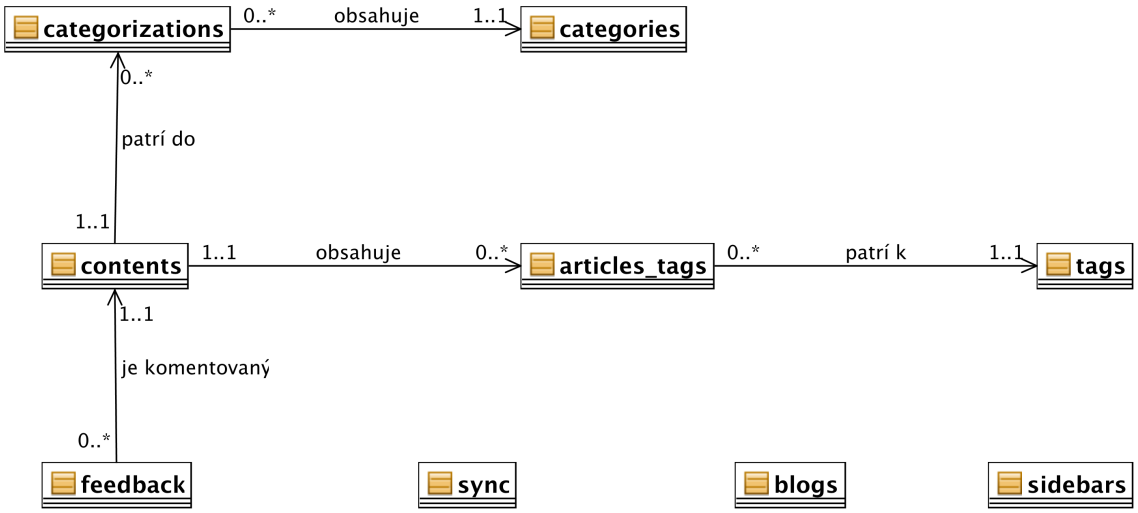
6.5 Návrh

Model návrhu ďalej upresňuje model analýzy vo svetle skutočného implementačného prostredia. Model návrhu tak predstavuje abstrakciu zdrojového kódu, inými slovami povedané, reprezentuje výkresovú dokumentáciu určujúcu ako bude zdrojový kód štrukturovaný a napísaný [25].

Mnoho súčasných webových aplikácií využíva viacvrstvový návrh systému. Medzi najznámejšie viacvrstvové architektúry patrí architektúra trojvrstvová, ktorá umožňuje oddelenie prezentačnej vrstvy, vrstvy aplikačnej logiky a dátovej vrstvy. Hlavnou výhodou takéhoto rozdelenia je možnosť implementácie jednotlivých vrstiev nezávisle



Obr. 5: Triedny diagram



Obr. 6: Dátový model

od ostatných. V našom návrhu implementácie budeme vychádzať práve z tejto trojvrstvovej architektúry a to konkrétne z architektúry Model View Controller.

6.5.1 Model View Controller

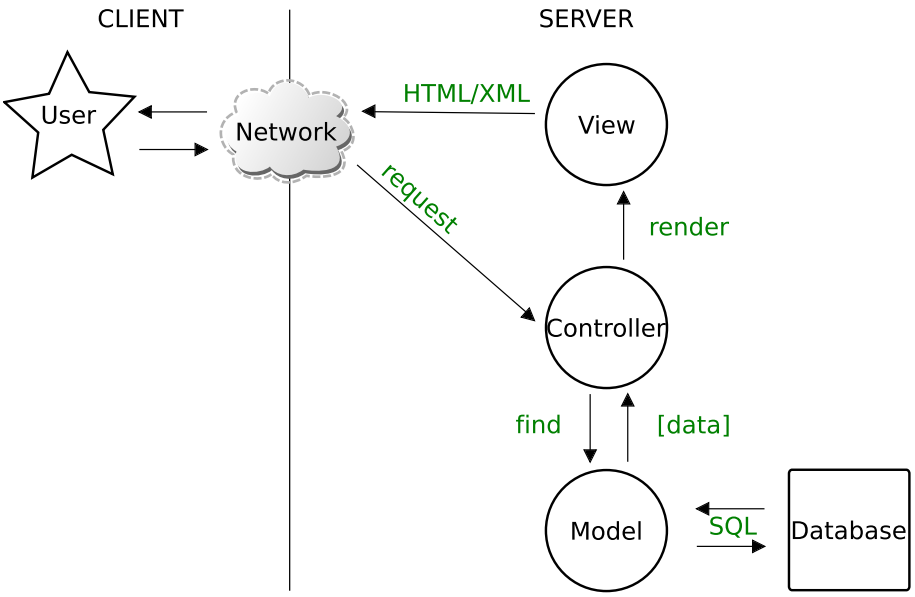
Je jeden z najpoužívanejších architektonických vzorov v oblasti tvorby webových stránok . MVC rozdeľuje architektúru systému do troch nezávislých komponent tak, aby modifikácia jednej z nich mala minimálny vplyv na ostatné.

6.5.1.1 Model Model je funkčným a dátovým základom celej aplikácie. Poskytuje prostriedky pre prístup k dátovej základni a stavom aplikácie, ale aj pre ich ukladanie a aktualizáciu. Predstavuje logiku celej aplikácie a zodpovedá napríklad za činnosti ako je ukladanie, aktualizácia dát, výpočty a iné.

6.5.1.2 View Zobrazuje výstup modelu a zaisťuje grafický alebo iný výstup aplikácie. Prostredníctvom modelu pristupuje k dátam a špecifikuje, ako majú byť tieto dáta zobrazené. V prípade webových stránok sa najčastejšie jedná o výstup do HTML.

6.5.1.3 Controller Controller definuje chovanie aplikácie. Spracováva vstupy a udalosti pochádzajúce od užívateľa. Na ich základe volá príslušné procesy Modelu, mení jeho stav a pod. Podľa vstupov od užívateľa, ale i podľa výsledkov akcie v Modely, Controller vyberá vhodné View pre ďalšie zobrazenie.

Pri návrhu výslednej architektúry sa musíme ešte zamyslieť nad problémom, akým spôsobom budeme implementovať našu offline aplikáciu v rámci už existujúcej aplikácie Typo. Poďme si ukázať, akým spôsobom je realizovaná súčasná aplikácia Typo. Architektúru aplikácie môžeme vidieť na obrázku 7.



Obr. 7: Pôvodná architektúra MVC aplikácie Typo

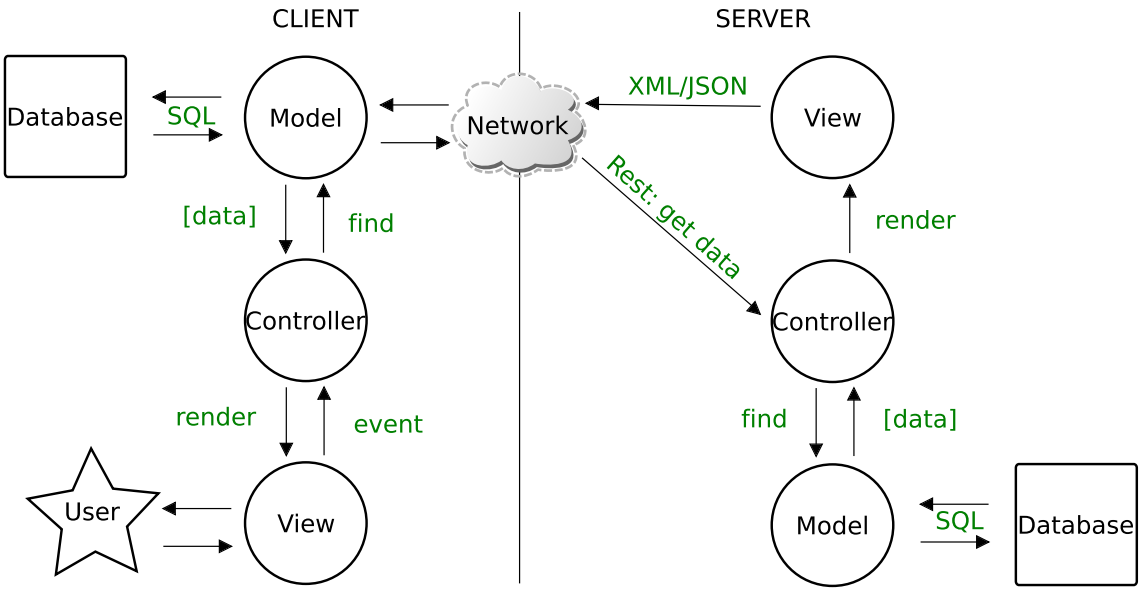
Aplikácia Typo používa taktiež návrhový vzor Model View Controller. Avšak ako môžeme pozorovať z obrázku 7, internetový prehliadač na strane užívateľa slúži ako tenký klient, to znamená, že celá logika aplikácie a vykonávanie jednotlivých operácií prebieha na strane servera a prehliadač len zobrazuje výsledky týchto operácií. V protiklade s týmto prístupom je offline aplikácia, ktorá musí poskytovať funkcionalitu aj v prípade nedostupnosti internetového pripojenia, to znamená aj v prípade nedostupnosti servera. Z tohto dôvodu musíme offline aplikáciu implementovať takým spôsobom, aby sme tákéto správanie zabezpečili. Navrhovaným riešením môže byť presunutie časti aplikácie spoločne s databázou na stranu klienta. V takomto prípade by internetový prehliadač neslúžil len ako tenký klient, ale časť logiky a vykonávanie jednotlivých operácií by prebiehali na strane klienta. Na obrázku 8 môžeme vidieť upravenú architektúru aplikácie, ktorá spĺňa požiadavky, ktoré sme si definovali pre správnu funkčnosť offline aplikácie.

6.6 Implementácia

V tejto časti si predstavíme niektoré dôležité aspekty systému z pohľadu jeho implementácie.

6.6.1 Offline mód

Základom celej implementácie offline časti aplikácie Typo je manifest súbor, ktorý sme si predstavili v kapitole 4.7. Hlavným cieľom manifest súboru je udržiavať zoznam URL adries súborov, ktoré si internetový prehliadač ukladá do svojej internej pamäte a



Obr. 8: Navrhovaná architektúra MVC aplikácie Typo

tieto súbory následne zobrazuje užívateľovi. V tomto zozname sa nachádza kompletný zoznam súborov, ktoré sú potrebné pre správne fungovanie aplikácie. Medzi tieto súbory patria javascript súbory vývojového rámca JavascriptMVC, HTML súbory, css štýly a obrázky. V zozname sa taktiež nachádza špeciálna sekcia nazvaná NETWORK, ktorá udržuje zoznam súborov, ktoré sa nemajú načítavať z internej pamäte prehliadača, ale prostredníctvom internetu. Je to z dôvodu, aby sa pri synchronizácii načítavali vždy aktuálne dáta. Zdrojový kód manifest súboru môžeme vidieť na výpise zdrojového kódu 42.

```
CACHE MANIFEST
#version 7.1.5

/ offline /typosphere.html
/ offline /jmvc/include.js?typosphere,production
/ offline /apps/typosphere/production.js
/ offline /resources/themes/ims/stylesheets/application.css
/ offline /resources/themes/ims/stylesheets/reset.css

#jquery ui
/ offline /resources/jquery-ui-1-2/css/ui-lightness/jquery-ui-1.7.2.custom.css
/ offline /resources/jquery-ui-1-2/css/ui-lightness/images/ui-bg_diagonals-
    thick_20_666666_40x40.png
/ offline /resources/jquery-ui-1-2/css/ui-lightness/images/ui-bg_gloss-
    wave_35_f6a828_500x100.png
/ offline /resources/jquery-ui-1-2/css/ui-lightness/images/ui-bg_glass_100_f6f6f6_1x400.png
/ offline /resources/jquery-ui-1-2/css/ui-lightness/images/ui-bg_glass_100_fdf5ce_1x400.png
```

```
/ offline /resources/jquery-ui-1-2/css/ui-lightness/images/ui-bg_highlight-
    soft_100_eeeeee_1x100.png
/ offline /resources/jquery-ui-1-2/css/ui-lightness/images/ui-icons_ffffff_256x240.png


#theme
/ offline /resources/themes/ims/images/menu-selector.gif
/ offline /resources/themes/ims/images/date-bg.png
/ offline /resources/themes/ims/images/search-button.gif
/ offline /resources/themes/ims/images/wrapper-bg.png
/ offline /resources/themes/ims/images/content-bg.gif
/ offline /resources/themes/ims/images/body-bg-2.jpg
/ offline /resources/themes/ims/images/rss.png
/ offline /resources/themes/ims/images/footer-bg.gif
/ offline /resources/themes/ims/images/box-header-bg.gif
/ offline /resources/themes/ims/images/social/rss.png
/ offline /resources/themes/ims/images/social/digg.png
/ offline /resources/themes/ims/images/social/facebook.png
/ offline /resources/themes/ims/images/social/delicious.png

NETWORK:
/sync/blog.json
/sync/contents.json
/sync/categories.json
/sync/categorizations.json
/sync/feedback.json
/sync/sidebars.json
/sync/tags.json
/sync/ articles_tags .json
```

Výpis 42: Kód manifest súboru v offline aplikácii

6.6.2 Prezentačná vrstva

Prezentačná vrstva v JavascriptMVC je riešená šablónami EJS, ktoré predstavujú kombináciu HTML kódu a špeciálnych značiek, ktoré umožňujú rýchle a pohodlné zobrazovanie prezentovaných informácií. Tieto šablóny sú svojou syntaxou a funkcionalitou veľmi podobné ERB šablónam z Ruby on Rails. Medzi základné funkcie šablón patria funkcie pre zobrazenie dát, iteračné funkcie alebo funkcie, ktoré umožňujú používanie šablón v rámci ostatných šablón. Ďalšou dôležitou funkciou je možnosť oddeliť hlavnú šablónu stránky od jednotlivých zobrazovaných častí. Prezentačná vrstva taktiež poskytuje mechanizmus vytvárania pomocných metód (anglické označenie helpers), ktoré predstavujú komponenty obsahujúce prezentačnú logiku zdieľanú medzi ostatnými šablónami. Zdrojový kód zobrazenia zoznamu článkov v JavascriptMVC môžeme vidieť na výpise zdrojového kódu 43.

```
<section>
<h1>List of articles</h1>
    <% if(! articles .length) { %>
```

```
        No articles
    <% } %>
    <% for(var i = 0; i < articles.length ; i++) { %>
        <%= view('views/article/show',articles[i]) %>
    <%}%>
</section>
```

Výpis 43: Zoznam článkov pomocou EJS šablóny

6.6.3 Riadiaca vrstva

Riadiacu vrstvu predstavuje časť aplikácie nazvaná controller. Trieda controller obsahuje obsluhu jednotlivých udalostí. Obsluhy udalostí sú metódy, ktoré spracovávajú udalosti pochádzajúcich z časti view. Spracovanie takýchto udalostí predstavuje vytvorenie metódy a namapovanie udalosti na takto vytvorenú metódu. Controller teda predstavuje prostredníka medzi prezentačnou vrstvou a vrstvou dátovou. Zdrojový kód triedy CategoryController, ktorá obsluhuje udalosť zobrazenia článkov vybranej kategórie, môžeme vidieť na výpise zdrojového kódu 44.

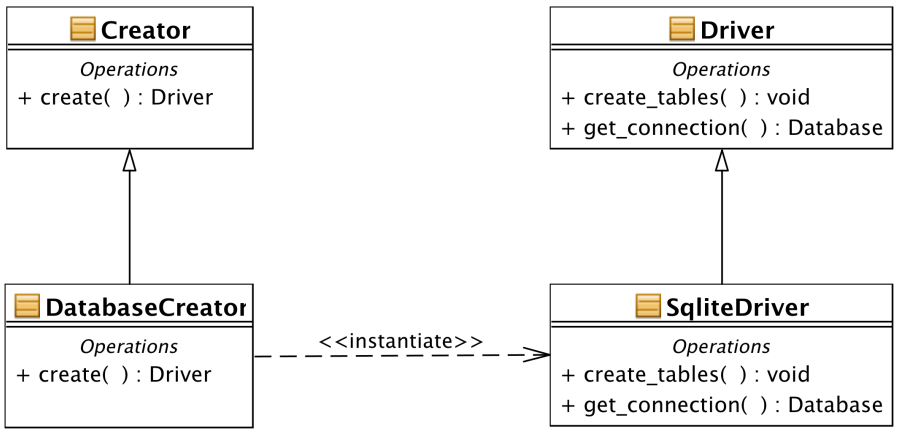
```
jQuery.Controller.extend('CategoryController',
/* @Static */
{
    onDocument: true
},
/* @Prototype */
{
    show: function(category,params) {
        $('.section').html(this.view('init', {
            category:category,
            params:params,
        }))
    },

    // Events handlers

    "history.category.index_subscribe": function(called, data) {
        var category = data.view
        var page = (data.page != null && data.page > 0) ? data.page : 1

        Category.find_by_permalink({
            current_page:page,
            category:category
        }, this.callback('show'), this.callback( Notification.msg))
    },
});
```

Výpis 44: Trieda CategoryController a zachytenie udalosti zobrazenia kategórie



Obr. 9: Návrh dátovej vrstvy

6.6.4 Dátová vrstva

K dátam v offline aplikácii pristupujeme prostredníctvom modelu, ktorý ich získava z relačnej databázy fungujúcej na strane klienta. Takúto databázu predstavuje SQLite relačná databáza.

6.6.4.1 Návrh dátovej vrstvy V rámci dátovej vrstvy nášho systému musíme vyčleniť triedy, ktoré sa budú starať o naviazanie spojenia s databázou. Tieto triedy musíme navrhnúť takým spôsobom, aby bolo možné zmenou konfigurácie aplikácie určiť, aký typ databázového serveru bude naša aplikácia používať. K tomuto účelu využijeme návrhový vzor Factory method.

Návrhový vzor Factory method patrí do skupiny takzvaných tvoriacich návrhových vzorov a jeho účelom je definovať rozhranie pre vytváranie objektov, pričom to, o akú inštanciu triedy sa jedná, rozhoduje odpovedajúca podtrieda [26].

Hlavným dôvodom pre použitie tohto návrhového vzoru je ten, že samotná špecifikácia HTML5 neuvádza žiadny typ databázového serveru, ktorý by považovala za štandardný pre špecifikáciu Web Database API. Z tohto dôvodu treba myslieť na možnú budúcu zmenu takéhoto databázového serveru. Výsledný návrh tried pomocou návrhového vzoru Factory method môžeme vidieť na obrázku 9.

6.6.4.2 Konfigurácia databázy Pred samotným získaním dát z databázy, je nevyhnutná prvotná inicializácia. Takáto inicializácia obsahuje okrem samotného naviazania spojenia s databázou, aj inicializáciu databázových tabuliek. Na strane klienta môže byť súbežne vytvorených niekoľko databáz od rôznych aplikácií, preto k identifikácii databázy používame identifikátory akými sú názov, verzia, veľkosť a popis. Na výpise zdrojových kódov 45 a 46 môžeme vidieť triedy, ktoré nám zabezpečujú naviazanie spojenia s databázou a následnú inicializáciu databázových tabuliek. Pre prehľadnosť výpisu zdrojového kódu je inicializácia databázových tabuliek obmedzená len na

vytvorenie tabuľky articles_tags.

```
jQuery.Class.extend("DatabaseDriver", {

    },
    {
        init : function(name, version, size, description) {

            // objects can only be created from DatabaseFactory
            if (this.Class.caller != DatabaseFactory.create) {
                throw new Error("There is no public constructor for " + this.Class.className)
                return
            }

            this.name = name
            this.version = version
            this.size = size
            this.description = description
            this.connection = openDatabase(this.name, this.version, this.description, this.size)
        },

        create_tables: function(queries) {
            var self = this
            jQuery.each(queries,function(i, val){
                self.connection.transaction(function(tx){
                    tx.executeSql(val)
                    })
            })
        },

        get_connection: function() {
            return this.connection
        },
    })
```

Výpis 45: Trieda DatabaseDriver a jej metódy pre prácu s databázou

```
DatabaseDriver.extend('SqliteDriver',
    {
    },
    {
        init : function(){
            this._super('SqliteTypo ','1.0','10485760',' Offline database for Typosphere')
        },
        create_tables: function() {
            var queries = [
            "CREATE TABLE IF NOT EXISTS articles_tags (article_id INTEGER, tag_id
                INTEGER)",
            ]
            this._super(queries)
        }
    });
```

Výpis 46: Trieda SqliteDriver a jej metódy pre prácu s databázou

6.6.4.3 Práca s databázou Na výpise zdrojového kódu 47 môžeme vidieť akým spôsobom prostredníctvom modelu získavame dáta z databázy.

```
$.Model.extend('Article',
/* @Static */
{
    find_page: function(page, success, error) {
        var self = this
        db.transaction(function(tx) {
            tx.executeSql("SELECT * FROM contents WHERE (contents.name = ?) AND ( (
                contents.type = 'Page') ) LIMIT 1", [page],
                function(tx, rs) {
                    return success(self.parse_result(rs))
                },function(tx,err){
                    return error(err)
                })
        })
    },
},
/* @Prototype */
{
})
```

Výpis 47: Trieda Article a získanie dát z databázy

6.6.5 Synchronizácia dát

Cieľom synchronizácie dát je zabezpečiť synchronizáciu databázy bežiacej na strane klienta s databázou bežiacou na strane servera. Synchronizácia je obojsmerná a samotný proces prebieha prostredníctvom RESTful webovej služby. REST predstavuje architektonický vzor pre distribuované systémy, akými sú napríklad WWW [27]. RESTful webové služby predstavujú webové služby implementované prostredníctvom HTTP a základných princípov REST. Takéto služby môžeme chápať ako kolekciu zdrojov s tromi definovanými aspektami:

- URI - základné URI webovej služby,
- MIME typ - typ dát podporovaných webovou službou (typicky JSON, XML alebo ostatné validné MIME typy),
- HTTP metódy - množina webových operácií podporovaných webovou službou, ktoré používajú HTTP metódy (napr. GET, PUT, POST a DELETE).

Princíp fungovania RESTful webovej služby môžeme zjednodušene zhrnúť do postupnosti nasledovných krokov. Užívateľ alebo systém na základe URL žiada webovú službu o poskytnutie prostriedkov (v tomto kontexte môžeme prostriedky chápať napríklad ako webovú stránku, obrázok, xml súbor alebo iné). V našom prípade by mohla požadovaná URL predstavovať adresu http://www.example.com/article/10. Webová služba na základe obdržanej URL vyhodnotí, aké prostriedky sa pokúšame získať. V našom prípade webová služba vie, že sa jedná o článok s identifikátorom 10. Na základe obdržanej HTTP metódy vykoná operáciu nad požadovanými prostriedkami a posiela odpoveď v závislosti na vykonávanej operácii. Predstavme si, že sa jedná o HTTP metódu GET, ktorá znamená, že požadujeme článok s id 10 a ako odpoveď by sme obdržali práve tento článok. Ak by sa však napríklad jednalo o HTTP metódu PUT, webová služba by vyhodnotila, že požadujeme aktualizovať článok s id 10 a článok by aktualizovala. Pre ďalšie štúdium REST odporúčam dizertačnú prácu pôvodného autora REST Roy Fieldinga Architectural Styles and the Design of Network-based Software Architectures [27].

Na výpise kódu 48 môžeme vidieť implementáciu REST na strane klienta v Typo aplikácii. Jedná sa o implementáciu časti webovej služby, ktorá zabezpečuje prácu s komentármi.

```
class Sync::FeedbackController < Sync::BaseController
  #HTTP method GET
  def index
    @feedback = Feedback.find(:all)
    @feedback.each { |comment| comment.body = comment.html }
    respond_to do |format|
      format.json { render :json => @feedback.to_json(:only => [:id,:type,: title ,:author,:body,:
          excerpt,:created_at]) }
    end
  end

  #HTTP method POST
  def create
    @comment = Comment.new
    @comment.author = params[:author]
    @comment.body = params[:body]
    @comment.email = params[:email]
    @comment.url = params[:url]
    @comment.article_id = params[:article_id]

    if @comment.save
      render :text => "Comment has been saved", :status => 200
    else
      render :text => "Comment has not been saved. Please try again.", :status => 500
    end
  end
end
```

Výpis 48: Implementácia REST v aplikácii Typo

7 Implementácia sémantiky

Cieľom časti sémantika je implementovať nové elementy jazyka HTML5 do systému Typo a nahradiť nimi v súčasnosti používaný jazyk HTML 4. Výsledkom tejto implementácie bude nová grafická téma, ktorá bude obsahovať plne sémantický kód. Pred samotnou implementáciou musíme však najprv identifikovať významné časti systému, ktoré budú slúžiť ako základ pre našu implementáciu. Pri popise jednotlivých riešení vždy porovnáme implementáciu HTML5 s pôvodnou implementáciou.

7.1 Doctype, css štýly a charset

Prvou zmenou, ktorú budeme implementovať je zmena definovania doctype, css štýlov a charset kódovania. HTML5 v tejto oblasti prináša zjednodušenie definovania týchto deklarácií.

7.1.1 HTML4

Na výpise zdrojového kódu 49 môžeme vidieť kód jazyka HTML 4 v pôvodnej šablóne Typo.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http :// www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8" />
        <title>HTML5 blog</title>
        <link href="/stylesheets/theme/style.css" media="all" rel="stylesheet" type="text/css" />
    </head>
    <body></body>
</html>
```

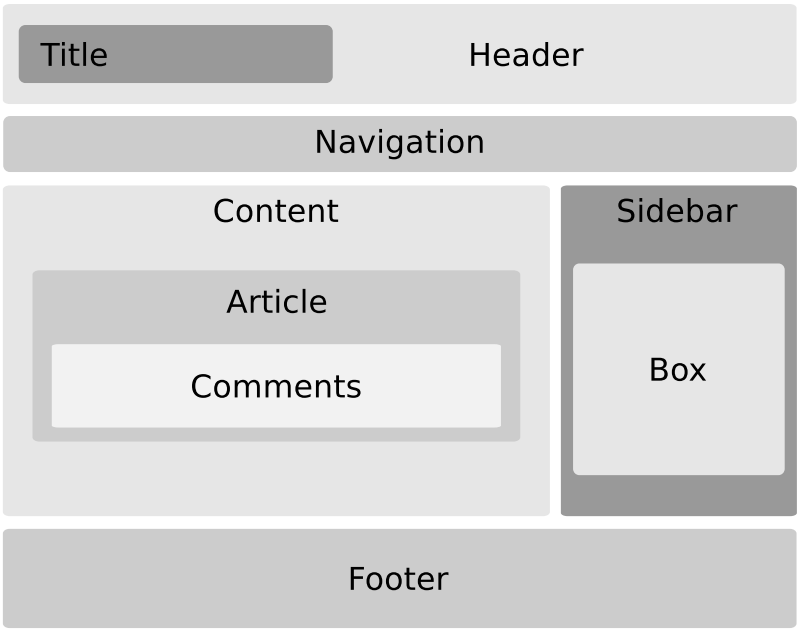
Výpis 49: Definovanie doctype, css a charset v HTML4

7.1.2 HTML5

Na výpise zdrojového kódu 50 môžeme vidieť kód jazyka HTML5 v novej šablóne Typo.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>HTML5 blog</title>
        <link href="/stylesheets/theme/style.css" media="all" rel="stylesheet" type="text/css">
    </head>
    <body></body>
</html>
```

Výpis 50: Definovanie doctype, css a charset v HTML5



Obr. 10: Spoločné časti systému Typo

Pri porovnaní výpisov zdrojových kódov jazyka HTML 4 a HTML5 môžeme konštatovať, že jazyk HTML5 skutočne prináša zjednodušenie syntaxe diskutovaných definícií.

7.2 Šablóna vzhľadu

Moderné webové stránky využívajú systém šablón. To znamená, že aplikácia dokáže zabaliť spoločné časti stránky akými sú menu, boxy, hlavička alebo pätička stránky do jedného súboru tzv. šablóny. Šablónu by sme mohli chápať ako kostru každej stránky, ktorá sa skladá zo spoločných grafických prvkov.

Typická šablóna systému Typo obsahuje následovné spoločné časti, ktoré môžeme vidieť na obrázku 10 [4].

7.2.1 HTML 4

Na výpise zdrojového kódu 51 môžeme vidieť kód pôvodnej šablóny systému Typo.

```
<body>
    <div id='container'>
        <div id='logo'>
            HEADER
```

[4] anglické názvy sú uvedené z dôvodu náväznosti na odpovedajúce HTML elementy

```
            <h1>TITLE</h1>
        </div>
        <div id='page'>
            <div id='content'>
                CONTENT
            </div>
            <div id='sidebar'>
                SIDEBAR
                <h3>BOX</h3>
            </div>
            <br class='clear' />
            <div id='legal'>
                FOOTER
            </div>
        </div>
        <ul id='menu'>
            NAVIGATION
            <li>Links</li>
        </ul>
    </div>
</body>
```

Výpis 51: Šablóna systému v HTML 4

7.2.2 HTML5

Na výpise zdrojového kódu 52 môžeme vidieť kód novej šablóny systému Typo.

```
<body>
    <header id="logo">
        HEADER
        <hgroup>
            <h1>TITLE</h1>
        </hgroup>
    </header>
    <div id="wrapper">
        <nav id="navigation">
            <h1>NAVIGATION</h1>
            <ul>
                <li>Links</li>
            </ul>
        </nav>
        <div id="content">
            <div id="content-inner">
                CONTENT
            </div>
            <aside id="boxes">
                <h1>SIDEBAR</h1>
                <section>
                    <h1>BOX</h1>
                </section>
            </aside>
```

```
        </div>
    </div>
    <footer id="footer">
        FOOTER
    </footer>
</body>
```

Výpis 52: Šablóna systému v HTML5

Pri porovnaní výpisov zdrojových kódov HTML 4 a HTML5 môžeme pozorovať, akým spôsobom sme vniesli význam pôvodnej šablóne systému. V prvom kroku sme si identifikovali spoločné prvky, ktoré sa vyskytujú v danej šablóne. V ďalšom kroku sme navrhli novú šablónu, ktorej cieľom bolo sémanticky označiť tieto identifikované prvky. Z výpisu zdrojového kódu 52 môžeme pozorovať, že pre časti header, footer a navigation máme odpovedajúce HTML elementy, ktorými sú <header>, <footer> respektíve <nav>. Z tohto pohľadu bol výber nových elementov jednoznačný. Ďalšími identifikovanými časťami sú sidebar, content a box, ku ktorým odpovedajúce elementy nemáme a preto sme museli zvoliť iné ekvivalentné elementy. Pre sidebar sme zvolili element <aside>. Definícia tohto elementu hovorí, že element aside predstavuje sekciu stránky, ktorá sa skladá z obsahu, ktorý priamo súvisí s obsahom okolo elementu aside, avšak nepredstavuje hlavný obsah. Keď sa zamyslíme, že časť sidebar zobrazuje informácie, ktoré priamo súvisia s obsahom stránky (napr. zoznam kategórií, zoznam nálepiek atď.), je nám jasné, že element aside je správnou voľbou. Pre content časť sme zvolili element a to z dôvodu, že vopred nevieme presne určiť, aký typ obsahu bude daná časť obsahovať. V tomto prípade sa tu naskytuje otázka použitia elementu <section>. V definícii tohto elementu sa okrem iného uvádza, že pokiaľ sekciu nie je možné prirodzene uviesť titulkom, použitie elementu section nie je vhodné. Keďže v našom prípade nepoznáme obsah tohto elementu, nedokážeme ani vopred určiť prirodzený titulok pre takýto obsah. Z tohto dôvodu sme sa rozhodli použiť element div na úkor elementu section. Poslednou identifikovanou časťou je časť box. Pre túto časť sme zvolili element <section>, ktorý predstavuje vhodného kandidáta pre daný druh obsahu. Definícia elementu section hovorí, že element môžeme chápať ako tématické zoskupenie obsahu, väčšinou uvádzané titulkom. Keďže boxy zobrazujú tématicky zoskupený obsah, element section predstavuje vhodného kandidáta.

7.3 Článok

Najdôležitejšou časťou každej webovej stránky je jej samotný obsah. Obsah systému Typo je zobrazovaný prostredníctvom článkov. Z tohto dôvodu je jednou z kľúčových úloh vniesť sémantiku do článkov.

7.3.1 HTML 4

Na výpise zdrojového kódu 53 môžeme vidieť pôvodný HTML kód článku.

```
<div class='article'>
    <h1>Article</h1>
    <p>Text</p>
    <div id='meta'>
        <p>Mon, 29 Mar 2010 15:17</p>
    </div>

    <h3 class="feedback">
        0 comments
    </h3>

    <ol id="commentList">
        <li>...</li>
    </ol>
</div>
```

Výpis 53: Kód článku v HTML 4

7.3.2 HTML5

Na výpise zdrojového kódu 54 môžeme vidieť nový HTML kód článku.

```
<article class="post">
    <header>
        <h1 class="title">Article</h1>
        <span>Posted by Tibor Pino, <time datetime="2010-03-11T11:09" pubdate>Date 11.
            03. 2010 11:09</time></span>
    </header>
    <div class="post-body">
        Text
    </div>
    <footer class="postmetadata">
        Posted in  Articles ,  1 comment
    </footer>
    <section class="comments">
        <h1>Comments</h1>
        <ol><li>...</li></ol>
    </section>
</article>
```

Výpis 54: Kód článku v HTML5

Pri porovnaní výpisov zdrojových kódov HTML 4 a HTML5 môžeme pozorovať, akým spôsobom sme vniesli význam článkom systému. V prvom kroku bolo potrebné sémanticky označiť článok ako celok. HTML5 nám ponúka pre označenie článku odpovedajúci element <article>. Ďalším krokom bolo rozdelenie článku na jednotlivé sekcie, akými sú hlavička, obsah článku, pätička a komentáre. Pre hlavičku sme zvolili odpovedajúci element <header>. Obsah tohto elementu tvoria základné informácie o článku akými sú názov, dátum a čas publikovania článku. HTML5 taktiež ponúka odpovedajúci element pre označenie časových dát. K označeniu dátumu a času publikovania článku

sme použili element <time>, ktorého voliteľný atribút *pubdate* označuje, že daný dátum a čas predstavuje čas publikovania článku. K označeniu obsahu článku nie je potrebný žiadny špeciálny element a preto sme použili element . Pre pätičku sme zvolili odpovedajúci element <footer>. Poslednou sekciou článku je sekcia komentáre. Pre túto časť sme zvolili odpovedajúci element <section>.

7.4 Formulár pre vloženie nového komentára

Ďalšou oblasťou, ktorú výrazne ovplyvňuje HTML5 je oblasť formulárov. HTML5 prináša niekoľko nových typov input elementov, ktoré otvárajú bohaté možnosti práce s formulármi.

7.4.1 HTML 4

Na výpise zdrojového kódu 55 môžeme vidieť pôvodný HTML kód formulára.

```
<form action="#" method="post">
    <div id='commentform'>
        <h3>Comment <em>article 2</em></h3>
        <p>
            <input id="comment_author" name="comment[author]" size="30" type="text" />
        </p>
        <p>
            <input id="comment_email" name="comment[email]" size="30" type="text" />
        </p>
        <p>
            <input id="comment_url" name="comment[url]" size="30" type="text" />
        </p>
        <p>
            <textarea cols="40" id="comment_body" name="comment[body]" rows="20"></
                textarea>
        </p>
        <p>
            <input type='image' id='form-submit-button' onclick='this.form.submit()' src='/
                images/theme/comment.jpg' />
        </p>
    </div>
</form>
```

Výpis 55: Kód formulára v HTML 4

7.4.2 HTML5

Na výpise zdrojového kódu 56 môžeme vidieť nový HTML kód formulára.

```
<form action="#" method="post">
    <div id="respond" class="post">
        <h2 class="title">Leave a comment</h2>
```

```
        <p>
            <input id="author" name="comment[author]" size="30" type="text" required>
        </p>
        <p>
            <input id="email" name="comment[email]" size="30" type="email">
        </p>
        <p>
            <input id="url" name="comment[url]" size="30" type="url">
        </p>
        <p>
            <textarea cols="40" id="body" name="comment[body]" rows="10"></textarea>
        </p>
        <p>
            <input id="form-submit-button" name="commit" type="submit" value="Submit">
        </p>
    </div>
</form>
```

Výpis 56: Kód formulára v HTML5

Pri porovnaní zdrojových kódov HTML 4 a HTML5 môžeme pozorovať, akým spôsobom sme vniesli význam jednotlivým prvkom formulára. V našom prípade sme použili nové typy elementov url a email. V prípade input elementu pre vloženie mena užívateľa sme použili jeden z nových atribútov *required*, ktorým sme určili, že daný element pri odoslaní formulára nesmie byť prázdny a musí obsahovať nejaké dáta. Zo zavedenia nových typov input elementov plynú mnohé výhody. V jazyku HTML 4 nebolo možné vo väčšom rozsahu určiť, aký typ informácií obsahuje daný input element (napr. pre získanie rôznych typov dát od užívateľa sa väčšinou používal input element typu text). Zavedením nových typov, ktoré dávajú input elementom bližší význam, môžu internetové prehliadače poskytovať rozšírenú funkcionalitu pri práci s takýmito elementami. Môžeme si predstaviť, že sa nám v prípade práce s elementom typu url na mobilnom zariadení automaticky zmení rozloženie klávesnice kvôli rýchlejšej a pohodlnejšej možnosti napísať internetovú adresu (klávesnica môže zobraziť tlačítko, ktoré bude obsahovať hodnoty .com .sk alebo .cz) alebo v prípade elementu typu email, kde nám môže byť zobrazené rozloženie klávesnice, ktoré bude obsahovať špeciálne znaky akými sú napríklad @. Možnosti v tomto smere sú veľmi široké a v blízkej budúcnosti sa určite budeme stretávať s väčšinou spomínanej funkcionality [5].

[5] Mobilné zariadenie iPhone už v súčasnosti podporuje možnosti práce s elementami typu url a email, a pri práci s nimi poskytuje špeciálne rozloženie virtuálnej klávesnice

8 Implementácia SVG

Cieľom implementácie SVG do systému Typo je vytvoriť takú grafickú interpretáciu systému, ktorá bude nezávislá na rozlíšení zariadenia, ktoré ju zobrazuje. Každá webová stránka sa skladá zo základných prvkov, prostredníctvom ktorých sa zobrazujú informácie. Takýmito prvkami sú napríklad text v kombinácii s HTML elementami alebo napríklad obrázky. Do tejto skupiny však patria aj grafické prvky, ktoré môžu byť deklarované v štýloch webových stránok alebo napríklad pomocou už spomínaných obrázkov. Problémom grafických prvkov, prostredníctvom ktorých zobrazujeme nejaké informácie je, že v prípade škálovania takýchto prvkov dochádza k strate zobrazovaných informácií. Preto našou prvotnou úlohou bude identifikovať prvky, ktoré predstavujú grafické prvky systému. Následne sa tieto grafické prvky budeme snažiť nahradiť prvkami vytvorenými pomocou SVG tak, aby v prípade škálovania takýchto prvkov nedochádzalo k strate zobrazovaných informácií. Ďalšou úlohou v procese implementácie SVG do systému Typo bude navrhnúť rozloženie týchto grafických prvkov takým spôsobom (môžeme chápať taktiež ako vytvorenie šablóny), aby toto rozloženie bolo konzistentné naprieč širokou škálou rôznych rozlíšení.

8.1 Grafické prvky

V sekcii 7 sme si identifikovali spoločné časti, ktoré tvoria šablónu systému. Výsledok môžeme vidieť na obrázku 10. Medzi spoločné časti patria title, header, navigation, content, sidebar, boxes a footer. Nie každá z týchto častí je reprezentovaná grafickým prvkom, preto sa pri implementácii SVG obmedzíme len na časti navigation, box a footer. Z ostatných prvkov, ktoré majú význam pri implementácii, a nie sú zobrazené na obrázku 10, sú tlačítka pre zobrazenie článku a pozadie samotnej stránky. Z identifikovaných grafických prvkov si bližšie predstavíme len prvky navigation a box, ktoré dostatočne demonštrujú použité princípy.

8.1.1 Navigation

Vytvorenie grafického prvku navigácie môžeme vidieť na výpise zdrojového kódu 57.

```
<svg id="svg-navigation" width="100%" height="3em">
    <rect width="100%" height="100%" rx="10" ry="10" x="0" y="0" style="fill:url(#blue_gray); fill
        -opacity:.8" />
    <g><a xlink:href="/">
            <text x="1em" y="1.8em">
                Home
            </text>
        </a></g>
    <g><a xlink:href="/archives">
            <text x="5em" y="1.8em">
                Archives
            </text>
        </a></g>
```

```
    <g><a xlink:href="/pages/about">
            <text x="10em" y="1.8em">
                About
            </text>
        </a></g>
</svg>
```

Výpis 57: Kód grafického prvku navigácia v SVG

Základným elementom pre vloženie SVG grafiky do HTML kódu je element <svg>. Následne sme v rámci tohto elementu vytvorili navigačný element prostredníctvom SVG elementu <rect>, ktorého obsah tvoria tri elementy, ktoré vytvárajú navigačné odkazy.

8.1.2 Box

Vytvorenie grafického prvku box môžeme vidieť na výpise zdrojového kódu 58

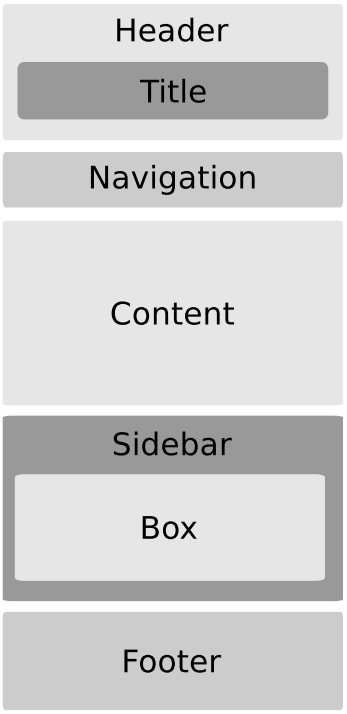
```
<section>
    <svg id="search" width="100%" height="2em">
        <g>
            <rect
                width="100%"
                height="100%"
                style=" fill :url(#blue_gray_small);" />
            <text x="1em" y="1.3em"><%= @sidebar.title %></text>
        </g>
    </svg>
    <form action="#" method="get">
        <p><input type="search" id="q" name="q" value="" size="15">
            <input type='submit' value='<%= _("Search") %>'></p>
    </form>
</section>
```

Výpis 58: Kód grafického prvku box v SVG

V tomto prípade sme použili kombináciu SVG a HTML5. Grafický prvok predstavuje názov boxu a jeho pozadie. Tento grafický prvok sme preto implementovali pomocou SVG elementov <rect> a <text>. Ostatné informácie, ktoré nepredstavujú grafické prvky, sú implementované pomocou HTML5.

8.2 Šablóna vzhľadu

Základnou požiadavkou na vytvorenie šablóny je konzistencia rozloženia grafických prvkov naprieč širokou škálou rôznych rozlíšení. Z tejto požiadavky vyplývajú ďalšie obmedzenia kladené na výslednú šablónu. Jedným z takýchto obmedzení je aj šírka šablóny. Keďže v našom prípade nedokážeme dopredu určiť minimálnu ani maximálnu šírku (z dôvodu rôznych rozlíšení zariadení, ktoré stránku zobrazujú), musíme rozloženie komponovať tak, aby nebolo závislé na žiadnej konkrétnej hodnote. Viac informácií o



Obr. 11: Navrhovaná šablóna v SVG

špecifických požiadavkách kladených na mobilné webové stránky sa môžete dozvedieť v dokumente 5 Can't-Miss Usability Tips for Mobile Website Designs [28].

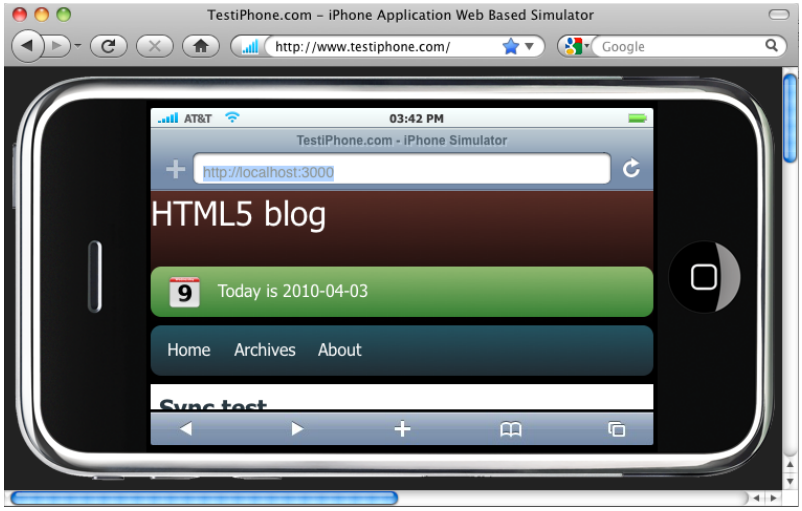
Vhodným riešením, ktoré spĺňa kladené obmedzenia, môže byť vertikálne rozloženie grafických prvkov radených postupne za sebou. Navrhované riešenie môžeme vidieť na obrázku 11.

SVG grafické prvky sú navrhnuté spôsobom, aby sa automaticky prispôsobovali šírke obrazovky a vypĺňali tak voľný priestor. Spoločne s navrhnutou šablónou vytvárajú výslednú verziu systému, ktorá je vždy konzistentná naprieč širokou škálou rozlíšení a grafické informácie sú reprezentované vždy v rovnakej kvalite bez straty zobrazovaných informácií.

V dobe písania diplomovej práce je jediným dostupným prehliadačom s podporou SVG a HTML5 Firefox 3.6 [6]. Z tohto dôvodu naše testovanie obmedzíme len na tento prehliadač. Pri testovaní budeme taktiež používať simulátor mobilného telefónu iPhone [29], ktorý nám umožní lepšiu simuláciu rozlíšenia obrazovky tohto zariadenia. Výsledné zobrazenie systému prostredníctvom tohto simulátora v prehliadači Firefox môžeme vidieť na obrázkoch 12 a 13.

[6] pre správne fungovanie musí mať Firefox explicitne nastavenú konfiguračnú premennú html5.enable na hodnotu true

Obr. 12: SVG zobrazenie v iPhone



Obr. 13: SVG zobrazenie v iPhone

9 Záver

Cieľom diplomovej práce bolo predstaviť nový štandard jazyka HTML s názvom HTML5 a implementovať vybrané časti tohto štandardu do existujúceho systému Typo. V prvotnej fáze som identifikoval problémy súčasného štandardu HTML 4 a webových stránok, a na základe týchto identifikovaných problémov som navrhol riešenia prostredníctvom implementácie vybraných častí HTML5. Medzi tieto časti patria implementácia offline módu, sémantiky a SVG.

Na základe získaných skúseností a poznatkov z procesu implementácie môžem prehlásiť, že HTML5 predstavuje štandard, ktorý v dostatočnej miere ponúka riešenia mnohých problémov, ktoré sú spojené so súčasnými webovými stránkami. Medzi takéto problémy napríklad patria práca s webovými stránkami v prípade nedostupnosti pripojenia, nedostatočná sémantika HTML kódu, video a audio alebo celkový spôsob práce s webovými stránkami alebo formulárami. HTML5 však pokrýva aj iné oblasti, ktoré sú pre bežného užívateľa skryté a takýmito oblasťami sú WebSockets (komunikačný kanál), WebWorkers (javascriptové vlákna) alebo mechanizmus postMessage (medziokenná komunikácia). Pri HTML5 som však narazil aj na niekoľko problémov, ktoré sú hlavne spojené s licenciami a so záujmami veľkých firiem. Takéto problémy sa týkajú napríklad elementov video a audio, kde nebolo možné určiť štandardný kódek pre používané video a audio. Výsledkom tohto sporu je odstránenie zmienky o štandardnom kódeku z HTML5 a rozhodnutie, ktorý kódek bude podporovaný, je ponechané na samotných výrobcoch internetových prehliadačov. Obdobná situácia je aj v prípade Web Database API, kde HTML5 neurčuje štandardný databázový server (i keď všetci výrobcovia doposiaľ implementovali podporu SQLite).

Celková práca na tejto diplomovej práci bola veľmi zaujímavá a to z dôvodu, že mi umožnila detailne sa zoznámiť s novými technológiami, predovšetkým HTML5, ale i s Ruby on Rails a JavascripMVC. Výsledok implementácie nám ukazuje, že HTML5 je technológia, ktorá je pripravená na produkčné použitie a opäť posúva možnosti, ktoré nám ponúkajú webové stránky, o kus ďalej. Podporu jednotlivých technológií HTML5 v internetových prehliadačoch môžeme vidieť v dokumente HTML 5 Demos and Examples [31]. Výsledná aplikácia je umiestnená na adrese http://typo.galileoagency.sk.

10 Literatúra

[1] Wikipedia javascript
dokument dostupný na URL http://en.wikipedia.org/wiki/javascript

[2] Wikipedia SQLite
dokument dostupný na URL http://en.wikipedia.org/wiki/Sqlite

[3] An Interview with the Creator of Ruby
dokument dostupný na URL http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html

[4] About Ruby
dokument dostupný na URL http://www.ruby-lang.org/en/about/

[5] Dave Thomas, David Heinemeier Hansson,*Agile Web Development With Rails*, The Pragmatic Programmers LLC., 2007

[6] Wikipedia Rich Internet application
dokument dostupný na URL http://en.wikipedia.org/wiki/Rich_Internet_Application

[7] Oficiálna stránka pracovnej skupiny WHATWG
dokument dostupný na URL http://www.whatwg.org/

[8] WHAT open mailing list announcement
dokument dostupný na URL http://lists.whatwg.org/htdig.cgi/whatwg-whatwg.org/2004-June/000005.html

[9] results of HTML 5 text, editor, name questions
dokument dostupný na URL http://lists.w3.org/Archives/Public/public-html/2007May/0909.html

[10] HTML5 Draft standard
dokument dostupný na URL http://dev.w3.org/html5/spec/Overview.html

[11] Mark Pilgrim, *Dive into HTML5*
dokument dostupný na URL http://diveintohtml5.org/

[12] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*
dokument dostupný na URL http://www.w3.org/TR/xml

[13] About Microformats
dokument dostupný na URL http://www.microformats.org/about

[14] HTML5 differences from HTML4
dokument dostupný na URL http://www.w3.org/TR/html5-diff/

[15] Lachlan Hunt, *Supporting New Elements in IE*
dokument dostupný na URL http://blog.whatwg.org/supporting-new-elements-in-ie

[16] Web SQL Database specification
dokument dostupný na URL http://dev.w3.org/html5/webdatabase/

[17] Web Storage specification
dokument dostupný na URL http://dev.w3.org/html5/webstorage/

[18] jQuery - official website
dokument dostupný na URL http://www.jquery.com/

[19] Typo - oficiálna stránka
dokument dostupný na URL http://www.typosphere.org/

[20] Radiant - oficiálna stránka
dokument dostupný na URL http://www.radiantcms.org/

[21] Mephisto - oficiálna stránka
dokument dostupný na URL http://www.mephistoblog.com/

[22] Wordpress - oficiálna stránka
dokument dostupný na URL http://www.wordpress.org/

[23] Drupal - oficiálna stránka
dokument dostupný na URL http://www.drupal.org/

[24] Joomla - oficiálna stránka
dokument dostupný na URL http://www.joomla.org/

[25] Ivo Vondrák, *Úvod do softwarového inženírství*, 2004

[26] Ivo Vondrák, *Metody specifikace softwarových systému*, 2004

[27] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, 2000
dokument dostupný na URL http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm

[28] 5 Can't-Miss Usability Tips for Mobile Website Designs
dokument dostupný na URL http://spyrestudios.com/usability-tips-for-mobile-website-designs/

[29] TestiPhone.com - iPhone Simulator
dokument dostupný na URL http://www.testiphone.com/

[30] Wikipedia Drag-and-drop
dokument dostupný na URL http://en.wikipedia.org/wiki/Drag-and-drop

[31] HTML 5 Demos and Examples
dokument dostupný na URL http://html5demos.com/

[32] Michal Radecký *Sémantický web*
dokument dostupný na URL http://www.cs.vsb.cz/Files/osobni_stranky/michal-radecky/IT2009_3.pdf

A Štruktúra systému Typo

Systém Typo má zložitú adresárovú štruktúru, z tohto dôvodu je potrebné uviesť, v ktorých adresároch v rámci aplikácie sa nachádzajú jednotlivé implementované časti. Štruktúra je uvedená v tabuľke 1.

Offline aplikácia	$public/offline/$
	$app/controllers/manifest\_controller.rb$
	$app/views/manifest/$
REST	$app/controllers/sync/$
	$config/routes.rb$
Sémantika	$themes/html5/$
SVG	$themes/svg - mobile$

Tabuľka 1: Štruktúra implementovaných častí v Typo

B Štruktúra priloženého CD

Na priloženom CD nájdete implementáciu systému pre správu obsahu Typo vrátane konfiguračných a iných nevyhnutných súborov. Štruktúra CD je uvedená v tabuľke 2.

typo	v tomto adresári sú umiestnené zdrojové kódy aplikácie
docs	v tomto adresári je umiestnená dokumentácia
text	v tomto adresári je umiestnený text diplomovej práce

Tabuľka 2: Štruktúra priloženého CD